Implementación de Artificial Bee Colony para Controlador Bilineal en actuadores SMA

TRABAJO FIN DE GRADO

U. Carlos III de Madrid
Grado en ingeniería electrónica industrial y automática
Realizado por Zhisheng Ye Fu
Tutor: Fernando Martín Monar

AGRADECIMIENTOS

Quiero agradecer a todas las personas que me han acompañado a lo largo de este maravillo curso, mis familiares, amigos y profesores. Una mención especial al departamento de Ingeniería de Sistemas el cual me han enseñado y he adquirido, conocimientos y pasión por la ingeniería de control.

RESUMEN

Este trabajo consiste en la implementación del algoritmo ABC (*Artificial Bee Colony*) para buscar las mejores ganancias de un controlador PID Bilineal para actuadores SMA (*Shape memory alloy*). A continuación se realiza una breve descripción de los puntos más importantes de este trabajo.

El algoritmo que voy a utilizar en este trabajo se denomina algoritmo ABC. Sus siglas traducidas al español significan *colonia artificial de abejas*. Como su nombre indica, su desarrollo se basa en el comportamiento de búsqueda de alimento de las abejas comunes. Es un algoritmo que se ha desarrollado recientemente y su uso no es tan común, aunque en estos últimos años muchos investigadores han empezado a utilizarlo por su precisión y fiabilidad.

El controlador PID bilineal consiste en un regulador clásico PID al cual se le añade un componente bilineal. Este tipo de reguladores se usa en control de sistemas no lineales para compensar la no-linealidad de una planta. De este modo, facilita el modelado y el control de una función de transferencia no lineal de un sistema, que en el caso que aquí ocupa va a ser un actuador SMA. De acuerdo a esto, las ganancias que debe encontrar el algoritmo ABC para el controlador son: *Kp, Ki, Kd* (por parte del PID) y *Kb* (por parte del bilineal).

El actuador SMA (*Shape Memory Alloy*) es un dispositivo compuesto por una aleación metálica de níquel y titanio que se caracteriza por la propiedad de que puede "recordar" y volver a su forma original tras una deformación. Hasta el momento es un material no muy conocido y su uso no está tan extendido, aunque se espera que en un futuro cercano esté implementado en los aparatos de los distintos sectores, como el biomédico o el aeronáutico.

Los resultados experimentales muestran que se ha implementado satisfactoriamente el algoritmo y el optimizador es capaz de encontrar parámetros adecuados para el control bilineal cuando este se aplica para un modelo de planta que simula el comportamiento de actuador SMA.

ABSTRACT

This work consists of the implementation of the ABC (Artificial Bee Colony) algorithm in order to optimize the parameters of a Bilinear PID controller for SMA (Shape Memory Alloy) actuators. A brief description about the most important aspects of the project is given in the following paragraphs.

The algorithm that is the core of the optimization method is the ABC. It is based on the behavior of the common bees when searching for food. This technique is quite new and not very common among researchers. However, in the last years, its use has been increased due to its good performance regarding accuracy and reliability.

The Bilinear PID controller is based on a classic PID regulator with a bilinear term. This type of controller is used in non-linear system to linearize the non-linear plant. After that, it is easier to model and control the linearized system. In this case, the non-linear model is a SMA actuator. The gains to be optimized by the ABC-based technique are Kp, Ki, Kd (PID) and Kb (bilinear term).

The SMA actuator is a device formed by a metal alloy of nickel and titanium which is characterized by the property that it can "remember" and return to its original shape after deformation. The use of this material is not widespread, although it is expected that in the near future it will be implemented in devices of different sectors such as biomedical and aeronautics.

The experimental results show that the algorithm has been successfully implemented and the optimizer computes adequate parameters for the bilinear controller when it is applied to a plant model that simulates the behavior of a SMA actuator

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS Y TABLAS

1 INTRODUCCIÓN

Desde tiempos remotos, nosotros, los humanos, siempre hemos intentado mejorar nuestra calidad de vida y de trabajo, ingeniándolas para minimizar el esfuerzo y el riesgo de lesionarnos.

Otra cualidad nuestra es intentar siempre de alcanzar lo inalcanzable, aprovechando los materiales de nuestro entorno para fabricar y construir, y hacer realidad sueños e ideas que solo estaba en nuestra mente. Numerosos son los ejemplos, como la elaboración de porcelanas, el manejo de diferentes metales, la mezcla de distintos compuestos para la aplicación en la construcción o la creación de diversos polímeros etc.

A medida que se logra una maravilla tecnológica se abre otra meta a superar. De este modo, descubrimiento tras descubrimiento, experimento tras experimento, hemos llegado a la sociedad actual. Capaz de viajar grandes distancias en poco tiempo, llegar al espacio exterior, sanarnos de enfermedades peligrosas, crear oportunidades e independencia a las personas con falta de algún miembro, entre otras muchas.

Como se ha podido ver en la historia, cada sector se ha encontrado con alguna que otra restricción hasta el punto que se han visto obligado a paralizar el desarrollo de algún prototipo, pero que finalmente han salido hacia adelante por los nuevos descubrimientos y avances tecnológicos, un ejemplo es el caso de la visión por computador, la cual fue estudiada en los setenta pero se tuvo que dejar de lado por la poca capacidad que tenía los ordenadores de la época y siendo retomada una década después por el desarrollo de potentes microprocesadores. Otro buen ejemplo en el sector de la robótica son los actuadores que necesita este, que a día de hoy el más utilizado es el actuadores eléctricos, que en cierto modo son muy eficientes, pero limita mucho al robot por su tamaño y peso.

Es aquí, en este documento, de que se habla de un posible sustituto, el actuador SMA[1] (Shape memory alloy), aunque también se esté experimentado con otros actuadores más pequeños y ligeros que el eléctrico.

[1] Aleaciones con memoria de forma, es la traducción al español. Son metales que pueden recuperar su forma original tras la deformación de esta al aplicarle un estímulo dado, que suele ser térmico.

1.1 MOTIVACIÓN

Hoy en día no es descabellado pensar sobre un posible futuro el cual los robots estén presentes en nuestra sociedad y, que en gran medida, estos nos ayuden a realizar trabajos que para nosotros serían complicados. Pero hasta que llegue ese día debemos mejor todavía muchos aspectos, como la durabilidad de las baterías, el tamaño y el peso de los actuadores, el sistema de control, etc.

Estas limitaciones no son solo problemas para el diseño de un robot, sino también para prótesis, manos y piernas robóticas, exoesqueletos y otras más.

Por tanto, en la actualidad se está desarrollando y mejorando poco a poco cada ámbito, por ejemplo, en mi caso, el poder controlar de forma adecuada el actuador SMA sería un avance bastante importante, ya que este es ligero y poco voluminoso. Y sería de gran utilidad en el diseño de los aparatos de rehabilitación, porque se sustituiría los actuadores eléctricos por actuadores SMA, u otra aplicación es en la implementación de esta en los trajes de los astronautas para facilitar su trabajo a la hora de manipular en el espacio exterior.

Por este motivo, he elegido realizar este tema como trabajo de fin de grado para tomar un primer contacto más cercano al ámbito de la robótica, y de este modo aprender mucho más acerca de los robots y poder así aportar mi pequeño grano de arena, ahora y en el futuro.

1.2 OBJETIVO

El objetivo principal de este trabajo es comprobar la efectividad del algoritmo ABC[2] (Artificial Bee Colony) a la hora de buscar las mejores ganancias (*Kp*, *Ki*, *Kd*, *Kb*) para nuestro controlador PID bilineal[3]. De esta forma, trabajar de una manera adecuada el actuador SMA. Para ello se ha estudiado con detalle el controlador PID bilineal, las características del actuador SMA y el algoritmo ABC. Además, se ha realizado numerosas pruebas y experimentos para dar con la mejor respuesta.

[2] Algoritmo que simula el comportamiento de alimentación de las abejas comunes. Desarrollada por Karaboga en 2005.
[3] Controlador basado en el regulador tradicional PID añadiéndole un compensador bilineal.

Por tanto este trabajo se va a desarrollar de la siguiente manera. En un principio se va a mencionar los trabajos existente en la cual están relacionados con este documento, a continuación se va a realizar una explicación detallada de los materiales SMA, después, las estrategias de control más utilizadas en la actualidad y las que tienen un futuro prometedor (como nuestro controlador bilineal) y posteriormente una descripción de los optimizadores, centrándonos por su puesto en el algoritmo ABC.

Con toda la teoría presentada y entendida se mostraran en los últimos capítulos los resultados de los experimentos y finalmente la conclusión.

Palabras claves: *controlador PID bilineal, ganancias del controlador, actuador SMA, Nitinol, optimización, algoritmo ABC.*

2 ESTADO DEL ARTE

2.1 INTRODUCCIÓN

En este apartado se van a mencionar los trabajos existentes que hay hasta el momento y que están relacionados con el presente documento. Se va a dividir este capítulo en dos partes, la primera se tratará sobre los métodos de control y la segunda sobre el uso de distintos algoritmos para optimizar los procesos de control, todos ellos, tanto el control como los algoritmos, están aplicados en actuadores SMA.

2.2 ESTRATEGIAS DE CONTROL APLICADAS AL SMA

El uso de SMA como actuadores está en proceso de investigación y se espera por parte de los investigadores grandes resultados para poder ser implementados en los futuros robots u otros aparatos. La aplicación de este tipo de material en el campo de los actuadores se debe a que este presenta unas cualidades de relación peso/potencia excepcionales, lo cual se explicará con detalle qué es el SMA en el apartado 3 de este trabajo.

Pero para poder aplicar el material como actuador, se debe primeramente conocer su **sistema dinámico** para luego poder controlar su movimiento a nuestro antojo. Hasta el momento sabemos que el SMA como actuador es **difícil de controlar**, ya que se observa que su ciclo de trabajo tiene forma de **histéresis**[4]. Por esta particularidad que presenta el material, se ha estado utilizando, por parte de los investigadores, distintas estrategias de control para compensar la histéresis y, por consiguiente, controlar el sistema.

Por ejemplo, en el trabajo de Yang *et al.* [1] ensaya con una viga en la que se la incorpora en la superficie un actuador SMA, y el control ante una entrada escalón que utilizaron para el modelo de primer orden que se diseñó, fueron dos métodos:

- **Controlador** tradicional realimentado **PID**. Experimentalmente presenta un buen tiempo de establecimiento, pero tiene una sobreoscilación alta.

[4] Efecto de no reversibilidad presentado en ciertos materiales. Es decir, el recorrido de ida y vuelta de un punto a otro no son equivalentes.

- Modelo de **control directa**. Mediante este control se observa que el tiempo de establecimiento es bastante alto.

Un trabajo bastante interesante realizado por Popov *et al.* [2] utiliza dos métodos para controlar el actuador SMA y comparar finalmente los resultados. El que mejor resultados presentaba fue el **control de modelo interno** (IMC, sus siglas en inglés) pues con este, permite modelar al SMA con las 4 funciones de transferencia que incorpora (una ganancia, una F.T. de primer orden, otra de segundo orden y un tiempo muerto). El otro controlador que se utilizó fue uno basado en el método de **Ziegler-Nichols**. Un antiguo alumno de nuestra universidad lo utilizó también para controlar una garra robótica [3], su trabajo consistía en la sintonización de un controlador PID mediante el método Ziegler-Nichols.

Otro método que se ha utilizado, ha sido el control **PI + anti-windup** en el trabajo de Asua *et al.* [4] El efecto windup surge cuando el término integral del controlador se vuelve muy grande, de manera que si se alcanza los límites prefijados del actuador y se sigue integrando, el error de la salida irá incrementando. Otro trabajo que realizaron los mismos autores [5] fue **incorporar una red neuronal**[5] para compensar la histéresis del actuador SMA. En la figura 1 se puede observar las características de los dos métodos que están mencionados aquí arriba.

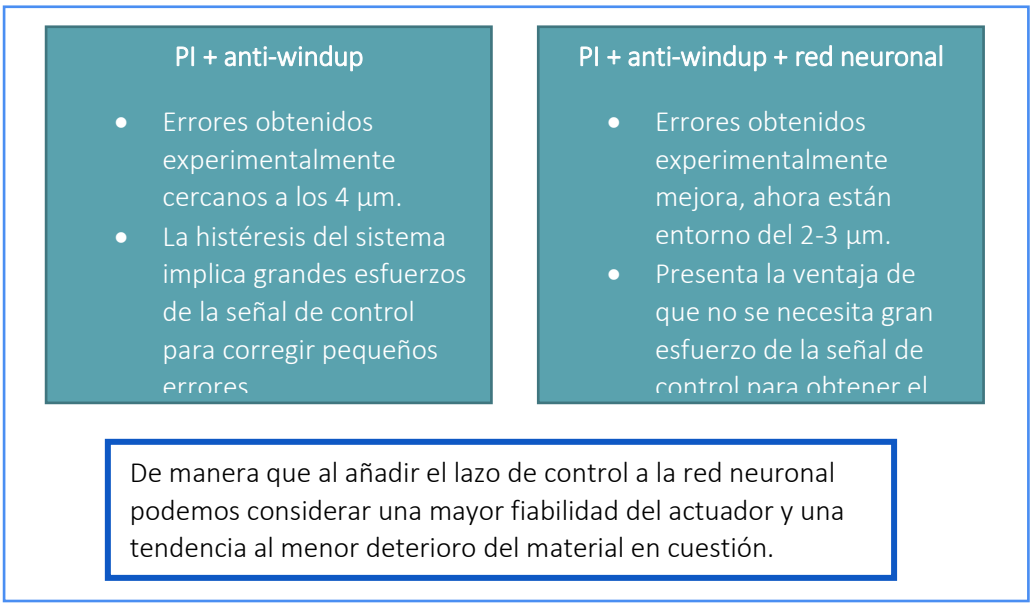


FIGURA 1: DIFERENCIA TRAS AÑADIR UNA RED NEURONAL AL CONTROLADOR PI + ANTI-WINDUP

En el siguiente trabajo realizado por Ma *et al.* [6] han obtenido un gran resultado al controlar el actuador SMA mediante el controlador **modulación por ancho de pulso** o

[5] En el apartado 4.2.5 se explicará lo básico de las redes neuronales.

PWM, las siglas en inglés. Este tipo de control difiere con los demás ya que en vez de trabajar en el dominio del tiempo continuo o periódico trabaja en tiempo discreto. Se ha demostrado que su uso es compatible con compensadores lineales y no-lineales. Y además se ha mostrado que reduce un buen porcentaje de consumo de energía por parte del actuador, en concreto, un 30% comparado con un controlador PD convencional. Por otra parte, una mejora de este tipo de control diseñado por los mismos autores [7], desarrollaron un controlador PD modulado por un modulador denominado Pulse-Width-Pulse-Frecuency, la cual presentaba un 50% menos de consumo por parte del SMA.

Actualmente se utiliza principalmente estrategias de control basadas en la combinación de controladores realimentados con controladores anticipativos o prealimentados. Como se puede comprobar, se obtienen buenos resultados, en los trabajos de Leang *et al.* [8] o de Kumagi *et al.* [9]. Y por último se está utilizando bastante el uso de controladores con ganancias programadas, la cual utiliza una serie de controladores lineales cuya actuación de estas es utilizada según en la situación en la que se encuentre la planta. La cual se puede encontrar este último en los trabajos de Jayender *et al.* [10] o de Kilicarslan *et al.* [11].

De modo que hasta el momento no se ha podido ver la aplicación del control PID bilineal en los actuadores SMA, y es por eso que en este trabajo se usará dicha estrategia.

2.3 ALGORITMOS DE OPTIMIZACIÓN EVOLUTIVOS APLICADOS EN CONTROL

Hasta el momento diversos investigadores han utilizado diferentes métodos de optimización. De las cuales que estén basados en algoritmos evolutivos y que estén aplicados en los controladores PID son los siguientes.

Los autores Chang y Chen [12] propusieron un **algoritmo genético adaptativo** para sintonizar los parámetros de un controlador PID fraccional, la cual se la aplicó para el control de sistema de soporte magnético activo.

El autor Zhang *et al.* [13] utilizaron como optimizador, de un controlador PID, un **algoritmo genético auto organizado**. Donde garantizaba una búsqueda global eficiente y presentaba una alta velocidad de convergencia.

En cambio Cao y Cao [14] apostaron en utilizar el **optimizador basado en enjambre de partículas** (PSO, las siglas en inglés) para la sintonización del orden fraccional del controlador PID.

Otro método realmente curioso, aplicado para la optimización de un controlador PID, es el que ha desarrollado los investigadores Korani *et al.* [15] consiguieron fusionar el concepto del **PSO con el comportamiento del desplazamiento de las bacterias E-coli**. Una mezcla realmente interesante.

Por último, recientemente, en el trabajo realizado Fernando Martín *et al.* [16] consiguieron realizan un control realmente bueno sobre un motor DC, mediante un controlador PID fraccionario y sintonizado con el método de **evolución diferencial** (DE, las siglas en inglés).

Visto los trabajos aquí arriba mencionados, la optimización busca el mejor resultado posible para el mejor funcionamiento de un sistema. Y hemos podido comprobar que existen numerosos métodos y tipos. Es aquí, en mi trabajo, que se utiliza el método de optimización basado en el comportamiento de las abeja comunes, para observar los resultados tras aplicarlo en sintonía con un controlador PID bilineal.

3 ACTUADORES SMA

3.1 INTRODUCCIÓN

Como se ha indicado, una de las desventajas existentes en el ámbito de la robótica es el tamaño y peso de los actuadores que tienen incorporados, por lo que en este apartado estará centrado en la explicación de un posible sustituto, los actuadores SMA.

3.2 GENERALIDADES. ¿QUÉ ES EL SMA?

Se definen materiales inteligentes los que son capaces de manifestar algún tipo de respuesta en función o presencia de estímulos exteriores, como ante una corriente eléctrica, campo magnético, etc. Y donde la respuesta producida por parte del material, pueda ser reversible, volver a su forma original (memoria de forma), que frecuentemente es de forma física (contracción y expansión, o de movimiento) o bien de otro tipo, como puede ser de cambio de resistencia. Todos estos materiales son, por ejemplo, los piezoeléctricos, las aleaciones de metales, aleaciones de polímeros…

De los materiales mencionados, se va a centrar en las aleaciones metálicas con memoria de forma, conocidas como SMA.

Este tipo de material, se conoce desde hace tiempo pero, hasta el momento, el que mejor cualidades presenta es la aleación Níquel-Titanio o Nitinol, para abreviar. Sobresale sobre el resto debido a su gran ductilidad, resistencia a la corrosión, capacidad de recuperación y biocompatibilidad. Por este motivo, se centrará en la descripción de este material.

La composición de esta aleación es equiatómico entre los dos metales y es bastante sensible respecto a este tema, ya que un exceso de 1% de Níquel introduce notables cambios, que generalmente incrementa la resistencia del material en su fase de austenita.

Actualmente existe una gran variedad de este tipo de aleación, con diferentes cualidades y dirigidas hacia diferentes aplicaciones específicas. Además según como sea el tratamiento mecánico y térmico en el proceso de fabricación puede presentar comportamientos distintos aun teniendo la misma composición de metales.

Este hecho hace que su fabricación para un rango de aplicación sea complicada para las empresas que se dedican a fabricarlas.

3.3 PRINCIPALES CARACTERISTICAS FÍSICAS DEL NITINOL

Tiene una capacidad de recuperación reversible de deformación hasta un 8,5%. Además su relación peso/potencia supera a cualquier otro actuador conocido. Por ejemplo, en el trabajo de Jonas Klemas [17] se observó que:

Un hilo de 10 cm de longitud, con 150 mm de diámetro tiene un peso total de 11,4 mg. Este es capaz de aplicar una fuerza de 8,8 N y contraerse hasta 0,85 cm. Es decir, es capaz de levantar un material que pesa 78.000 veces su propio peso casi 1 cm de altura.

Otra característica de la aleación Nitinol es que tiene un carácter no magnético, permitiendo así un uso más amplio.

3.4 FUNCIONAMIENTO DEL NITINOL

En concreto el Nitinol funciona en 3 estados:

1. Martensita, térmicamente inducido. En esta forma es blando y dúctil.
2. Martensita, súper-elástica. Obtenido por vía de tensión.
3. Austenita, forma estable en su rango de alta temperatura. En esta forma tiene propiedades de alta tenacidad y dureza.

El término de alta temperatura mencionado aquí arriba hace referencia al estado de estar por encima de una temperatura crítica que se la asocia a la temperatura cuando el material pasa de martensita a 100% austenita, que puede variar entre los -100º y los 200º de temperatura dependiendo del fabricante y el uso que se dé.

Si la martensita fue generada por vía de enfriamiento de una muestra del estado de austenita, tiene la propiedad de recuperar su formar original tras una deformación plástica mediante la aplicación de calor siempre y cuando supere la temperatura crítica. Este fenómeno se le conoce como memoria de forma.

En cambio, si la martensita fue inducida mediante una tensión que la deforma. Puede aparecer un material conocido como súper-elástico, si se supera ligeramente la temperatura en la que se convierte en 100% austenita. El cual, como se dice en el artículo [18], esto la hace maleable tanto en la fase de martensita como en la fase de austenita. Como se puede observar en la figura 2.

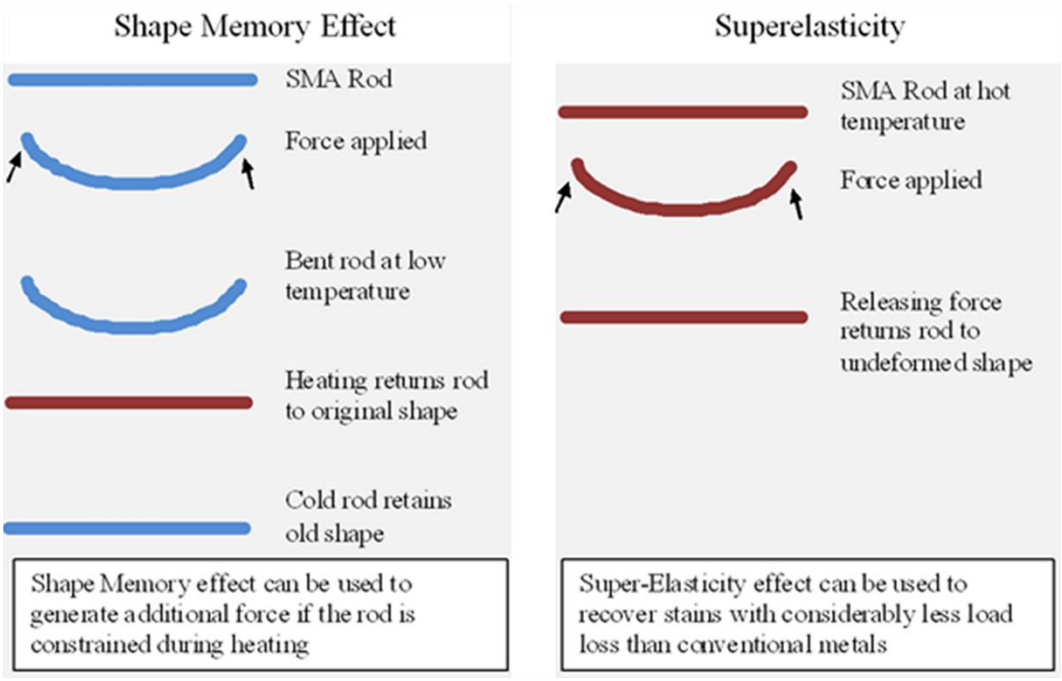


FIGURA 2: PROPIEDADES DE LOS MATERIALES SMA

Las barras en color azul están en estado martensita, y en rojo cuando supera la temperatura el cual es todo austenita [19]. Como se ha mencionado, solo en la fase de martensita el Nitinol es blando y dúctil, asimismo, cuando el material alcanza la propiedad de súper-elástico también es fácil de moldear en fase de austenita.

3.5 CARACTERISTICAS DEL FENÓMENO DE SÚPER-ELÁSTICO DEL NITINOL

Ya se ha insinuado anteriormente la aparición de martensita por vía de tensión. Para que se alcance con éxito este fenómeno, tiene que cumplir estas condiciones:

- Temperatura de tratamiento superior a la de la transformación a 100% austenita.
- Alcanzar una mínima tensión crítica según el tipo de aleación.

Todo esto bajo una operación isotérmica.

A diferencia de la propiedad de la memoria de forma, la súper-elasticidad tiene la cualidad de recuperar su forma con solo deformarla, alcanzando deformaciones de mayor magnitud y recuperables al 100% (puesto que en la memoria de forma tiene una deformación residual del 0,5%), pero se tiene que aplicar una mayor fuerza de deformación.

3.6 VENTAJAS Y DESVENTAJAS DE LOS ACTUADORES SMA

Las principales ventajas de los actuadores SMA, como se indican en los artículos de Leppäniemi [20] y Mavroidis [21], están en su uso en la robótica, ya que son de tamaño y peso reducido, el cual los hace muy codiciados para su aplicación en este ámbito. Sustituyendo de esta manera los pesados y complejos actuadores tradicionales.

El actuador SMA cuanta con una relación fuerza/peso muy alta, lo demostramos con el ejemplo que se mostró en el apartado ***principales características físicas del Nitinol.***

Hay que destacar que cuenta con un sistema sencillo, en el que consiste principalmente el uso de calor y enfriamiento, que puede ser accionada directamente por una corriente eléctrica y, donde se le puede repetir esta acción millones de veces sin que se pierda sus facultades. Comparando con los actuadores actuales es mucho más silencioso y no requiere de mecanismos de fricción, facilitando de esta manera su montaje (puesto que no necesita amplificadores ni de reductores), y su mantenimiento posterior. Y otra ventaja a señalar es que es un actuador relativamente barato, como se indica en este blog [22].

Además cuenta con una excelente resistencia a la corrosión y una alta biocompatibilidad.

De este modo, hace que los científicos e ingenieros tengan un especial interés en este tipo de material e intentar aprovechar estas cualidades en números campos. Como puede ser la biomédica (brazos y piernas robóticos, aparatos de rehabilitación, robots asistentes, etc.) o la aeronáutica (guantes y zapatos de los trajes para los astronautas).

Pero como todo, existen desventajas. En este caso, la eficiencia del actuador es pequeña, ya que al ser un sistema que usa principalmente el calor y enfriamiento del SMA, cumple con el ciclo de Carnot, y por tanto trabaja a una eficiencia entorno al 10%.

Asimismo, tiene un ancho de banda de operación pequeño, es decir, son lentos en completar el ciclo de trabajo, necesita un buen medio para disipar el calor. Pero si se mejora demasiado el ambiente de transferencia de calor se aumenta a la vez la energía necesaria para aplicar el calor. Por lo que hay que buscar un término en la cual compense las dos etapas de trabajo y que se realice de manera eficiente en ambos casos. Se debe mencionar que alcanza poca elongación al contraerse, solo un 8,5%, eso quiere decir que para realizar un movimiento de 10 cm necesitamos un actuador de más de 1 metro. También hay que tener cierta precaución con los sobrecalentamientos y las sobretensiones, ya que de esta manera se pueden degradar antes de lo esperado.

Otra desventaja es su **dificultad de ser controlado**, al ser un material donde su porcentaje de martensita en el compuesto es distinto cuando se le eleva la temperatura que al dejarlo enfriar, hace que su ciclo de trabajo tenga forma de **histéresis**, ver figura 3.

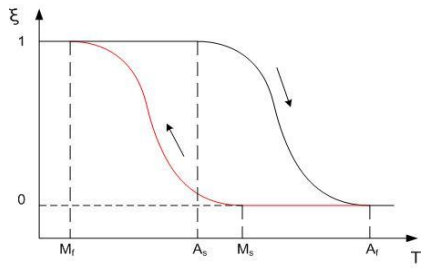


FIGURA 3: EFECTO HISTÉRESIS DEL SMA.

Este fenómeno aparece porque existe solo una orientación cristalográfica para formar la austenita y en cambio para la formación de la martensita hay 24 planos equivalentes. Y el desfase que sufre puede variar según el proceso de fabricación que ha tenido la aleación.

Es en este apartado donde se centra mi trabajo, buscar el mejor método y forma de controlar dicho actuador. Y el método utilizado es el control bilineal, el cual se explica a continuación.

4 ESTRATEGIAS DE CONTROL

4.1 INTRODUCCIÓN

En nuestro día a día, tenemos numerosos problemas a superar el cual recurrimos a nuestro ingenio y conocimiento para solucionarlos, y siempre buscamos alguna estrategia para llegar a la mejor solución. De la misma forma ocurre con los sistemas automáticos y se le conoce como estrategias de control [23].

Las estrategias de control determinan la estructura que sigue la información en el sistema, que dependiendo de la planta a controlar, los componentes que tiene y su finalidad estarán diseñados de una forma u otra. A continuación se va citar algunas estrategias de control típicas.

4.2 ESTRATEGIAS DE CONTROL COMUNES

En este apartado se va a citar las estrategias de control más utilizadas en las fábricas e industrias, como por ejemplo, puede ser: el control realimentado, control selectivo, control predictivo, control en cascada, control borroso, control PID o el uso de redes neuronales.

4.2.1 CONTROL REALIMENTADO

El control realimentado, o lazo cerrado, funciona de tal manera que hace que la salida del sistema vuelva al principio para que se analice la diferencia entre la entrada y salida ajustándose de tal manera que se intente que el error entre ambas sea nulo. Hay dos tipos de realimentación, la realimentación negativa

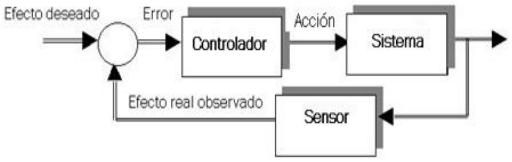


FIGURA 4: ESQUEMA CONTROL REALIMENTADO.

(más común, ayuda a mantener la estabilidad) y la positiva (asociada al incremento de posibilidades creativas). Se puede ver el esquema que sigue este tipo de control en la figura 4.

4.2.2 CONTROL EN CASCADA

Este controlador, ver figura 5, funciona de manera que la salida de un controlador realimentado es el punto de ajuste para otro controlador de realimentación, es decir, uno dentro de otro.

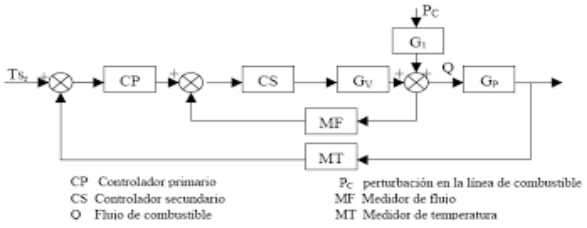


FIGURA 5: ESQUEMA CONTROL EN CASCADA.

Este método aporta las siguientes ventajas:

- Eliminar el efecto de algunas perturbaciones, de modo que el sistema se comportará de manera más estable y rápida.
- Mejorar la dinámica del lazo de control.
- Menos sensible a errores de modelado
- Incremento de la capacidad de la producción.

Algunos de los inconvenientes es que requiere medir las perturbaciones de forma explícita y en algunos casos la variable controlada no puede medirse y la realimentación no puede producirse.

4.2.3 CONTROLADORES BASADOS EN ESTRATEGIA PID

El controlador PID, ver figura 6, es uno de los más utilizado en el sector de la industria. Los motivos de su amplio uso son por su eficiencia y fiabilidad a la hora de controlar determinadas plantas o sistemas que hay en las fábricas, siempre y cuando la dinámica de este lo permita (normalmente cuando son de primer y segundo orden). Actualmente van acompañados con funciones lógicas y secuenciales para adecuarse a los requerimientos de los modernos sistemas de control permitiendo de esta manera automatizar dispositivos especializados para el control de temperatura, velocidad, etc.

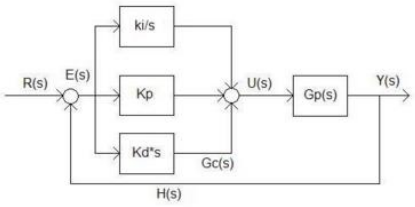


FIGURA 6: ESQUEMA CONTROLADOR PID.

La estructura del controlador PID es la combinación lineal de las acciones proporcional, integral y diferencial sobre el error generado por la planta o sistema. La combinación de estas hace incrementar el orden del sistema ya que introduce un polo en el origen y dos ceros. De esta manera permite que el controlador PID dé buenos resultados ante especificaciones de régimen permanente y transitorio, puesto que la acción integral elimina el error en régimen permanente y la acción diferencial mejora la respuesta del sistema en cuanto a la sobreoscilación y tiempo de respuesta.

	Kp aumenta	Ti disminuye	Td aumenta
Estabilidad	Se reduce	Disminuye	Aumenta
Velocidad	Aumenta	Aumenta	Aumenta
Error estacionario	No eliminado	Eliminado	No eliminado
Área del error	Se reduce	Disminuye hasta cierto punto	Se reduce
Perturbación control	Aumenta bruscamente	Aumenta gradualmente	Aumenta bruscamente
Frecuencia lazo	No afecta hasta cierto punto	Disminuye	Aumenta

FIGURA 7: TABLA DE PROPIEDADES DEL CONTROLADOR PID.

La acción de control PID tiene la siguiente forma:

$$u(t) = K(e(t) + \frac{1}{t_i}\int_0^T e(t)dt + t_d\frac{de(t)}{dt})$$

En la figura 7 indica las respuestas que se aprecian en una planta tras modificar los valores de las ganancias de cada parte del PID.

Nuestro sistema de control está basado en este modelo añadiéndole un enfoque bilineal, donde se explicará más adelante.

4.2.4 CONTROL BORROSO

El desarrollo del control borroso tiene origen en los *sistemas basados en reglas* para la toma de decisiones y en la *lógica borrosa* para la evaluación de estas, ver figura 8. La diferencia de este método es que no evalúa tablas ni ecuaciones, simplemente procesa los valores de entrada y deciden como modificar la salida para controlar la planta.

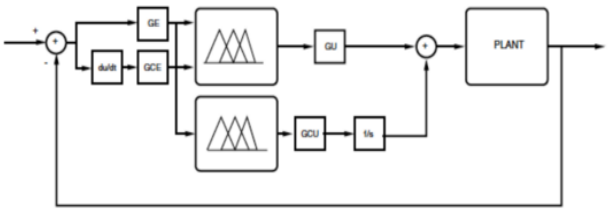


FIGURA 8: ESQUEMA CONTROL PD+I BORROSO.

De acuerdo con la información usada en el diseño podemos distinguir dos tipos de controladores borrosos:

- Controladores sin modelo. Estrategias que no se basan en modelos matemáticos, aunque necesitan información básica de algún modelo.
- Controladores basados en modelo. Usualmente requieren una descripción completa de la dinámica de la planta.

Un uso de este tipo de control es la de diseñarla como un tipo PID, utilizada para ajustar controladores lineales PID con modelos simples de respuesta. Su diseño consiste en dos controladores, uno equivalente al PD y otro suministra la acción integral, permitiendo así establecer un menor número de reglas para el controlador fuzzy.

De esta manera, mediante la actuación del controlador borroso y la aplicación de las ganancias adecuadas para cada parte (proporciona, integral y diferencial) se consigue controlar un sistema ante una entrada simple, como puede ser una entrada escalón.

4.2.5 CONTROL CON REDES NEURONALES

El desarrollo del sistema de redes neuronales artificiales, arranca en los años 40 del siglo XX, con los trabajos de McCulloch y Pitt que muestran que las redes neuronales artificiales podían calcular funciones aritmética y lógicas. Posteriormente se propuso mecanismos de aprendizaje para las neuronas. En los 50, Roseblatt demostró la capacidad de estas para el reconocimiento de patrones, más tarde en los 60 Widrow y Hoff introdujeron una nueva regla de aprendizaje que hasta el momento se sigue utilizando. Pero surge un parón en la década de los 70 debido a la ausencia de computadores sobre los que experimentar. Más tarde, con el desarrollo de los microprocesadores surgen nuevas ideas y posibilidades con las que ensayar, culminando con los trabajos realizados por Hopfield y más adelante por Rumelhart [24].

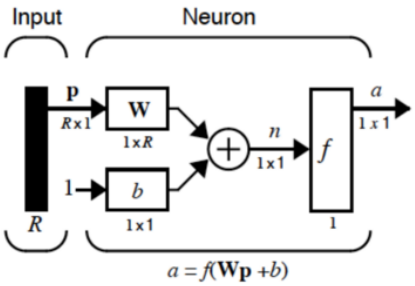


FIGURA 9: ESQUEMA DE UNA RED NEURONAL.

En la figura 9 se puede apreciar un modelo de una neurona artificial con varias entradas, que dependiendo de esta, saldrá una respuesta u otra.

Además, se puede complicar el modelo añadiendo más neuronas, o incluso diseñando más capas de neuronas, se muestra estos modelos respectivamente en las figura 10.

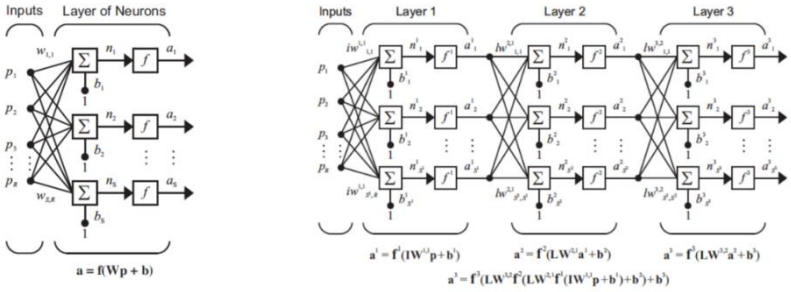


FIGURA 10: MODELOS CON MÁS NEURONAS (IZQ) Y CON MÁS CAPAS (DCH).

Antes de empezar a hablar del controlador con redes neuronales, vamos a recordar que cuando hablamos de control se puede distinguir dos grandes tipos:

- Regulación de sistemas. Se utiliza cuando hay que mantener una salida de un sistema en un cierto valor sin importar en las perturbaciones a las que está sometido.
- Servosistemas. Se busca que en un sistema tenga una salida que siga de la mejor forma posible una trayectoria de referencia dada. La cual en este apartado se va a centrar en este tipo.

Además, hay que tener en cuenta que la estructura del control en general está condicionada por la forma de identificar la dinámica inversa de la planta, por lo que se puede obtener lo siguiente:

- Modelo de control directo. Se utiliza cuando se consigue la información del sistema a controlar por método explícito de identificación, en otras palabras, son esquemas de control que necesitan un modelo del sistema para diseñar el controlador, por ejemplo: el control directo inverso o la linealización por realimentación.
- Modelo de control indirecto, figura 11. Se utiliza cuando la identificación de una planta forma parte de las acciones de control. Es decir, son esquemas de control que están basados en un modelo del sistema. La idea es en usar una red neuronal para modelar el sistema a controlar, por ejemplo: Posicionamiento de polos o el control predictivo.

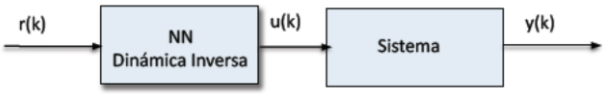


FIGURA 11: ESQUEMA MODE CONTROL INDIRECTO.

Ahora se puede hablar del control con redes neuronales, este controlador es un modelo de control directo inverso[6], cuya idea principal es conectar la red neuronal de forma paralela con el sistema para entrenarla, ver figura 12, con la dinámica inversa de la planta y así utilizarla luego como controlador.

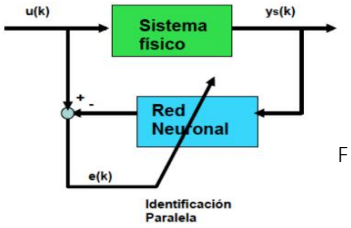


FIGURA 12: ESQUEMA DE ENTRENAMIENTO.

Si la red ha aprendido de forma exacta la dinámica inversa, la señal *u(k)* que genera conducirá al sistema a la salida deseada en *k+1*, tal y como se muestra en el esquema.

La figura 13 presenta ciertas ventajas y desventajas a la hora de entrenar para controlar un sistema.

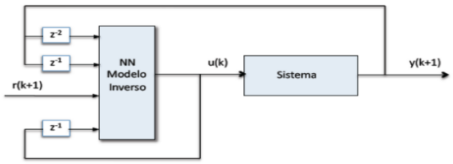


FIGURA 13: ESQUEMA CONTROL POR REDES NEURONALES.

VENTAJAS
Es intuitivamente simple.
Fácil de implementar.
Con el entrenamiento especializado, el controlador puede ser optimizado para una trayectoria específica.
Es posible su aplicación en sistemas variantes en el tiempo

DESVENTAJAS
No funciona en sistemas donde el inverso es inestable, suele ocurrir cuando se una muestreos a alta frecuencia.
Problemas con modelos inversos que no están bien amortiguados.
No hay opciones de ajuste.
Suelen ser altamente sensibles a

[6] Este modelo fue propuesto principalmente por Widrow, Albus, Miller.

4.3 CONTROL PID CON ENFOQUE BILINEAL

Como se ha mencionado en el apartado del control PID, nuestra estrategia de control utilizado para el actuador SMA se basa en un PID estándar el cual se le añade un sistema bilineal para facilitar el control de este. La cual mi estudio se basa en el trabajo de **Martineu *et al*** [25].

Sabemos que actualmente el PID está ampliamente usado en los procesos industriales y que proporciona una actuación satisfactoria y robusta, siempre y cuando se opere en el punto el cual mantiene la linealidad local. Pero como casi todos los sistemas que hay en nuestro entorno tienen un comportamiento no lineal, hace que su modelado y control sea complicado, del mismo modo hace que el control PID sea limitado. Ante este tipo de problemas se puede recurrir a las siguientes opciones:

- Buscar continuamente los parámetros de las ganancias del PID sobre los rangos de operaciones.
- Buscar los parámetros de las ganancias del PID el cual pueda habilitar el control del sistema para un rango de operaciones más ancho.

En la práctica, por restricciones de tiempo, disponibilidad personal y coste de puesta en marcha de la planta hace que la segunda opción sea la solución deseada. Para que la elección de esta opción tenga buenos resultados, se le debe modificar y/o añadir algún elemento al controlador PID, que en este caso va a ser un sistema bilineal.

Se ha podido comprobar que las estructuras bilineales suelen ser modelos, no-lineales, con fenómenos más precisos que las estructuras lineales. Y es por este motivo su aplicación en los sistemas no-lineales que hay en algunos de los procesos de las plantas industriales.

Las investigaciones sobre sistemas bilineales comienzan en la década de los 60 pero se intensificó en los 70 tras los progresos presentados por Mohler en su trabajo en modelados no-lineales para procesos biológicos y físicos, donde presentaba un ancho rango de trabajo para los sistemas mencionados.

Actualmente la CTAC (*Control Theory and Applications Centre*) de la Universidad de Coventry es la que está contribuyendo en gran medida la investigación y el desarrollo, y en consecuencia facilitando el desarrollo, del control basado en sistemas bilineales.

Para seguir hablando del control con enfoque bilineal (que lo llamaremos a partir de ahora *Control Bilineal*) vamos a introducir y explicar brevemente el concepto de sistemas bilineales.

Los sistemas bilineales representan una subclase dentro de los sistemas no lineales, el cual su ecuación dinámica sigue de la siguiente forma:

$$\dot{x}(t) = Ax(t) + bu(t) + u(t)Nx(t)$$

$$y(t) = C^T x(t)$$

Donde cada componente es:

- $x \in \mathbb{R}^n$ y representa el vector de estados,
- $u(t)$ es una entrada simple,
- A es una matriz de $n \times n$ compuesto por valores reales,
- b es un vector de $n \times 1$ compuesto de valores reales,
- N es una matriz de $n \times n$ compuesto de valores reales que contiene los coeficientes bilineales,
- $y(t)$ es la salida del sistema,
- C es un vector de $n \times 1$ compuesto de valores reales.

Ahora si recombinamos los términos de la ecuación se puede escribir de esta forma:

$$\dot{x}(t) = [A + u(t)N]x(t) + bu(t)$$

Se observa que tanto la respuesta dinámica como la respuesta del estado permanente dependen de la entrada del sistema. De este modo, para cualquier sistema bilineal con la forma de la ecuación superior, la representación del estado permanente de la salida ante una entrada también en estado permanente está expresada de la siguiente manera:

$$Y_{ss} = \frac{\beta_o * u_{ss}}{\alpha_o - \rho_o * u_{ss}}$$

De manera que la relación entrada-salida de la ecuación superior se puede ver en la figura 14.

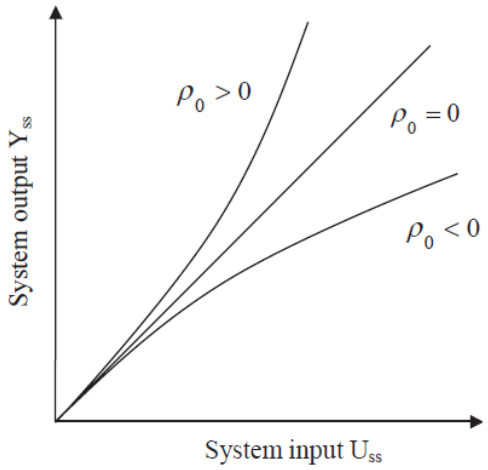


FIGURA 14: GRÁFICA RELACIÓN ENTRADA-SALIDA DE UN SISTEMA BILINEAL.

De la figura 14 podemos comprobar que si el término ρ_o es igual cero el comportamiento del sistema bilineal en régimen permanente es lineal. Por lo tanto, gracias a esta característica muchos desarrollos tecnológicos orientados a los sistemas lineales pueden ser extrapolados y aplicados en casos bilineales.

Hay que indicar que solo los sistemas bilineales con características idénticas a la gráfica superior son oportunos en adoptar un modelado y control bilineal.

Tras esta introducción y descripción general de los sistemas bilineales, se va a proseguir con la explicación del diseño de nuestro controlador PID bilineal.

Para resumir la explicación de nuestro controlador diremos que esencialmente es la combinación de un PID estándar con un sistema bilineal, que funciona para compensar la acción no lineal del SMA y poder así controlarlo. El controlador bilineal que se diseña sigue la estructura propuesta por Martineu *et al* [25], ya que presenta una ventaja, y es que el orden del modelo obtenido por el sistema bilineal es menor que al que correspondería para uno que se ha desarrollado mediante un sistema lineal. A continuación se muestra el sistema bilineal diseñada por Martineu para la aplicación en modelos de hornos industriales:

$$y\,(k) = \, -a * y(k-1) + b * u(k-1) + \, \eta * u(k-1) * y(k-1)$$

La ecuación representa la salida de la planta en el estado del dominio del tiempo discreto, la entrada viene dada por la expresión *u* y aporta la información en el tiempo *(k-1)*, por último se menciona que *a, b y* η son valores constantes.

Aunque este diseño sea creado para su implementación en los hornos industriales, se comprobará, en más adelante, que se puede utilizar perfectamente en los cables del actuador SMA.

Para el control de este tipo de no-linealidad se puede aplicar fácilmente una transformada algebraica que hace que la ecuación superior sea prácticamente lineal, el método más común y sencillo será la aplicación de la realimentación de estado. El cual consiste en realimentar el proceso del sistema provocando así una nueva entrada al proceso, y que está será capaz de controlar el sistema. Por tanto lo primero que se va a hacer es despejar la entrada de la ecuación inicial.

$$y(k) = \, -a * y(k-1) + \left(b + \, \eta * y(k-1)\right) * u(k-1)$$

Al surgir una nueva entrada por realimentación, está vendrá expresada de la siguiente manera en relación con la entrada original, como se hizo en el trabajo de Álvaro Villoslada *et al* [26].

$$u(k-1) = \frac{1}{1 + \frac{\eta}{b} * y(k-1)} * v(k-1)$$

La cual, si despejamos la constante b de la ecuación despejada y se define que:

$$Kb = \frac{\eta}{b}$$

Se podrá escribir de la siguiente manera:

$$y(k) = \, -a * y(k-1) + b * [1 + \, Kb * y(k-1)] * u(k-1)$$

Ahora se sustituye la relación de entradas obtenida tras la aplicación de la realimentación en la ecuación superior dando lugar lo siguiente.

$$y(k) = \, -a * y(k-1) + b * v(k-1)$$

Se concluye que en este momento el sistema es lineal y su control es posible con solo usar un controlador PID estándar. Pero debemos añadirle un último componente para garantizar que la actuación se mantiene en los puntos deseados, por tanto la expresión del modelo bilineal queda de la forma:

$$y\,(k) = \frac{1 + \ k_b \ * y_{ref}(k)}{1 + \ k_b * y(k-1)}$$

Siendo $y_{ref}(k)$ la salida de referencia para nuestro controlador.

Quedando la estructura del controlador PID bilineal como la de la figura 15.

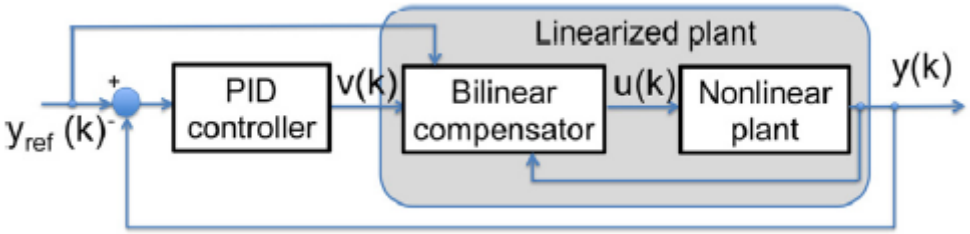


FIGURA 15: ESQUEMA CONTROLADOR BILINEAL.

De esta manera, se deberá hallar cuatro ganancias en vez de las tres del clásico PID, que son *Kp, Ki, Kd y* la necesaria para el control bilineal *Kb*.

5 OPTIMIZACIÓN. ALGORITMO ABC Y SU APLICACIÓN A CONTROL

5.1 INTRODUCCIÓN. BUSQUEDA DE LAS GANANCIAS DEL CONTROLADOR BILINEAL PID

Como se ha mencionado, el controlador diseñado necesita 4 ganancias para poder controlar el actuador SMA. Para ello podemos recurrir en proyectar en *Simulink* (programa de entorno visual que está incluido en *Matlab*) el modelo de la planta controlada siguiendo el esquema de figura 15 e introducirle valores de manera aleatoria a las ganancias y observar su comportamiento. Tras varias pruebas y con lo que se conoce de las características de las partes proporcional, integral y diferencial del PID (explicados en el apartado controladores basados en estrategia PID) se puede acabar encontrando los resultados deseados.

Otra manera de encontrar las ganancias deseadas, y de una forma más óptimas que utilizar solo la intuición, es manejar algún tipo de algoritmo de optimización que te ayude a buscar el mejor resultado. A continuación se va a realizar una breve introducción del concepto de optimización, seguidamente se mencionará de los algoritmos más comunes, el cual se hará una breve presentación y por último se va a hablar del algoritmo que se ha elegido para buscar los mejores valores de las ganancias.

5.2 OPTIMIZACIÓN

El término de optimización en la ingeniería se refiere al proceso de encontrar el mejor valor posible para un conjunto de variables mientras satisface varias condiciones. Según lo que se esté buscando por el diseñador, el término *mejor valor* tendrá un resultado u

otro. En resumidas cuentas, la optimización sirve esencialmente para mejorar el rendimiento de una actividad o proceso, evitando así la pérdida de tiempo y datos.

5.3 EJEMPLOS DE ALGORITMOS DE OPTIMIZACIÓN

En los siguientes apartados se presenta algunos algoritmos de optimización que se ha visto durante el grado y otros que en cierto modo tiene relación con el algoritmo elegido para el trabajo, ya que se basa en comportamientos observados en la naturaleza, como puede ser el comportamiento de las hormigas.

5.3.1 ALGORITMO GENÉTICO

El algoritmo genético es el método más conocido y usado de los métodos basados en estrategias evolutivos. Fue propuesto por D. Goldberg, J. Hallond, K. DeJong en 1989. Su funcionamiento básico es la siguiente, un conjunto de puntos que forma la población (S) evoluciona de una generación a otra mediante el reemplazo sucesivo de subconjuntos.

En los algoritmos genéticos, toda solución (x) es codificada por lo general usando una representación mediante una cadena binaria parecida a la forma en la que se codifican los cromosomas. Los tres conceptos básicos de un algoritmo genético son:

- Evaluación
- Selección
- Reproducción

La cual aplicando estos conceptos para crear el algoritmo genético como método de optimización se observa las siguientes características:

- No es muy rápido. Se codifican las soluciones potenciales en los cromosomas y se les aplica operaciones genéticas a los mismos. Estos son representados por las poblaciones, es decir, no trabajan con los objetos en sí, sino consiste en soluciones codificadas en problemas.
- Buena capacidad para resolver problemas combinatorios. Mide la “salud” en cada miembro de la población para indicar su nivel de adaptación.

- Enfatiza la combinación de información entre buenos progenitores. La combinación de información de ciertas poblaciones o progenitores son usados por los operadores de selección, cruce y mutación. Por este motivo el proceso es lento ya que se aplica estas operaciones en todas las generaciones.
- Existen muchas formas de diseñar un programa de algoritmo genético. Se afirma que una representación ética es importante en el éxito del algoritmo, ya que se ha demostrado que facilita el resolver del problema.

Con las descripciones de arriba se puede diseñar dos tipos de algoritmos genéticos:

- Simple.
- Canónica.

Del diseño simple del algoritmo podemos decir que sigue el siguiente esquema.

- Primero se genera una población aleatoria.
- Repetir este punto las veces que se quiera y/o se necesite.
    - Medición de la "salud" de cada integrante de la población.
    - Selección de los progenitores más aptos mediante métodos como el de la ruleta[7], basada en el rango[8] o el de torneo[9].
    - Cruce de los progenitores seleccionados mediante técnicas como el multi-punto o uniforme.
    - Mutación de la nueva población con métodos basados en las distribuciones gaussianas o series de Cauchy.
    - Reemplazar o inserción los progenitores por la nueva población mutada.

[7] Método en la cual se calcula en valor porcentual de la "salud" de cada miembro de la población mediante la siguiente ecuación: $p_i = \frac{f_i}{\Sigma_{i=1}^{P} f_i}$ el inconveniente de este método es que a veces se precipita en centrarse en valor "bueno" y descartar zonas que podrían contener resultados potencialmente buenos.

[8] Método en la cual se calcula la "salud" de cada miembro de la población y se ordena según su valor. Eligiendo de esta manera al mejor, que suele ser el primero.

[9] Método en la cual se elige aleatoriamente individuos que compite para ser elegidos como progenitores, donde el valor de la "salud" es el condicionante de su elección.

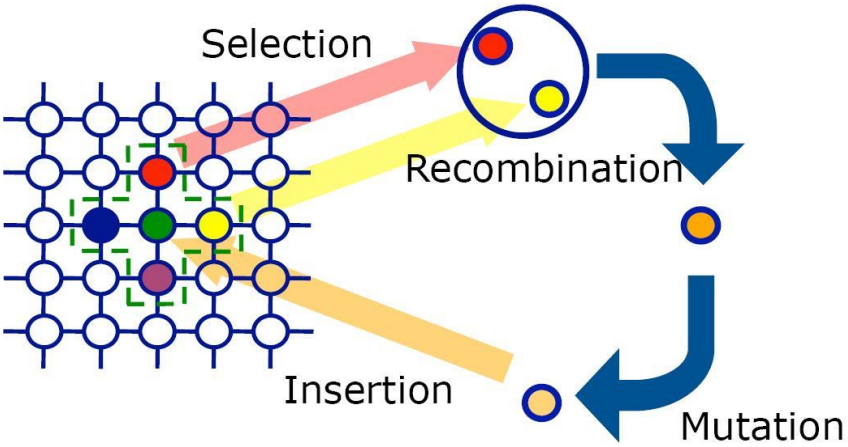


FIGURA 16: ESQUEMA DEL FUNCIONAMIENTO DEL ALGORITMO GENÉTICO.

5.3.2 OPTIMIZACIÓN POR COLONIA DE HORMIGAS

Es una técnica probabilística para solucionar problemas computacionales que pueden reducirse a buscar las mejores rutas en grafos. Forma parte del conjunto de algoritmos denominados inteligencia de enjambre[10] o del inglés *Swarm Intelligence*. Como su nombre indica, está basada en el comportamiento de las hormigas cuando estas están buscando un camino entre la colonia y una fuente de alimentos. Este comportamiento, tras numerosas observaciones, se ha concluido al siguiente patrón:

- Una primera hormiga encuentra la fuente de alimentos y retorna a la colonia por un camino cualquiera dejando un rastro de feromonas.
- Las demás hormigas se dirigen a la fuente de alimento, guiados por el rastro de las feromonas, aunque no tiene porqué ser el mismo camino, ya que si el camino es largo, las feromonas se evaporan antes dejando inexistente el camino.
- Entonces, solo el camino más corto entre la colonia y la fuente de alimento mantiene fuertemente el rastro de feromonas, puesto que pasan más hormigas sobre ella y dejan más feromonas, concluyendo de esta manera el hallazgo del camino más óptimo.

[10] Rama de la inteligencia artificial que estudia el comportamiento colectivo de los sistemas auto-organizados. Concepto introducido en el año 1989 e inspirados especialmente por los sistemas biológicos. [20]

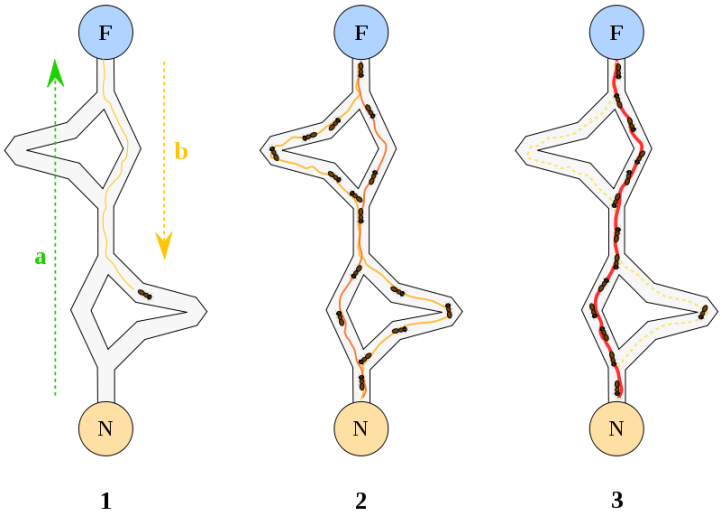


FIGURA 17: ESQUEMA DEL COMPORTAMIENTO DE LAS HORMIGAS.

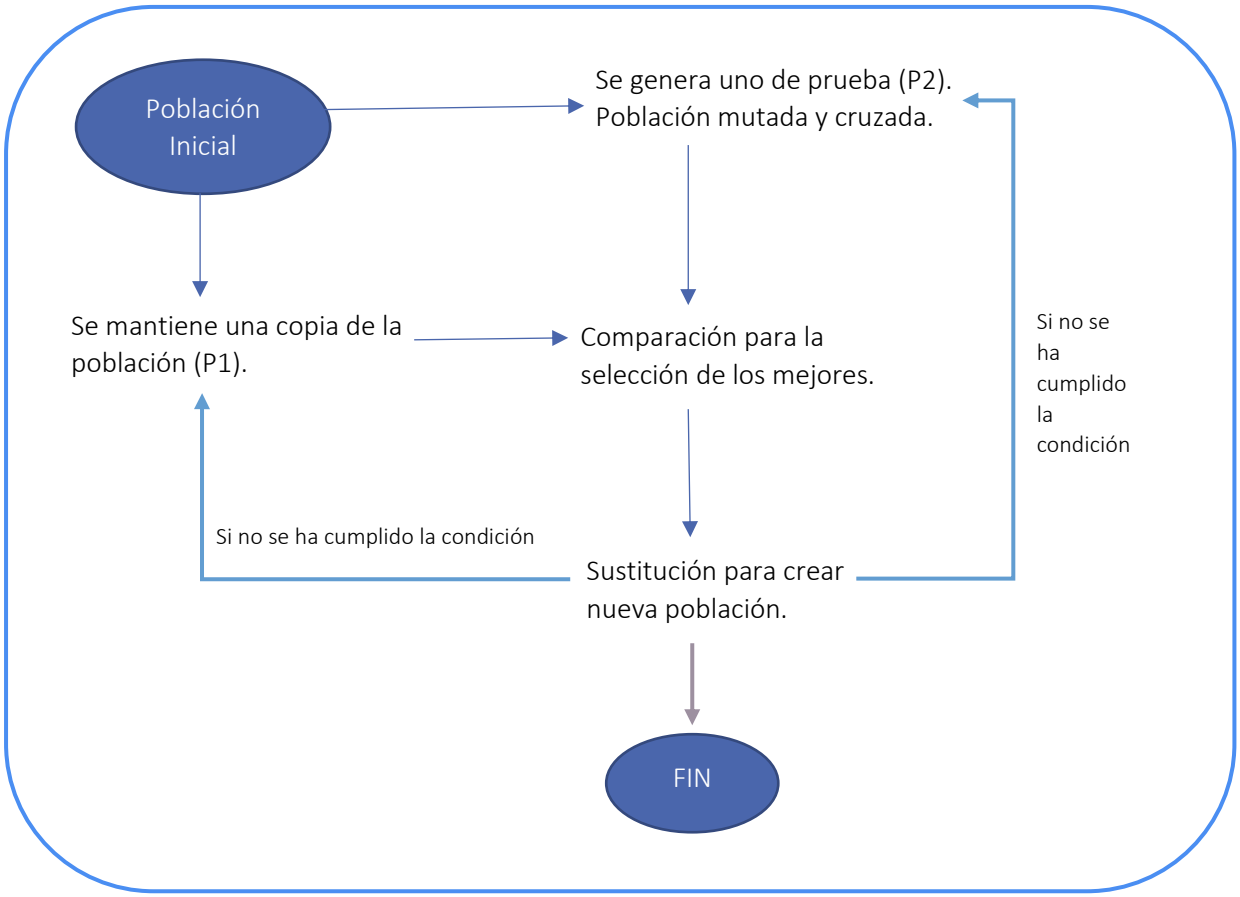
De éste método de optimización han surgido numerosos derivados que han sido adaptados a problemas dinámicos en variables reales, problemas estocásticos y programación en paralelo.

5.3.3 DIFFERENTIAL EVOLUTION

La primera aparición de este término surge en la publicación del artículo realizado por Kenneth Price y Rainer Storn en 1995 [27]. Y en un origen, el método, estaba enfocado en la resolución del problema de ajuste de polinomio de Chebyshev[11]. En el diagrama 1 que se muestra a continuación un esquema general del funcionamiento del algoritmo.

[11] Los polinomios de Chebyshev son una sucesión de polinomios ortogonales que están relacionados con la fórmula de De Moivre y son definidos de forma recursiva con facilidad. El uso de este polinomio está muy extendido en la teoría de la aproximación, porque las raíces de estos polinomios son usados como nodos en interpolación polinómica, permitiendo una mejor aproximación a una función continua bajo la norma maximal.

DIAGRAMA 1: DIAGRAMA DE FLUJO DEL FUNCIONAMIENTO DEL MÉTODO DIFFERENTIAL EVOLUTION.



Una vez presentado la idea principal, se debe mencionar las características y las cuestiones a tener en cuenta a la hora de diseñar el optimizador.

- Durante la inicialización hay que especificar los límites superiores e inferiores de cada parámetro. Por tanto cada miembro de la población tendrá un valor comprendido entre los límites.
- En el proceso de mutación se utiliza la siguiente expresión para crear un miembro de la población de prueba:

$$U = x_3 + F * (x_1 - x_2)$$

Donde las *x* son valores aleatorios de la población inicial y la *F* se la conoce como factor de escala, un valor comprendido entre 0 y 1.

- En el momento del cruce, se utiliza la estrategia del cruce uniforme entre la copia de la población y la mutada. Para determinar que fuente contribuye a un parámetro dado, se compara *Cr* (índice de probabilidad de cruce) con un generador de números aleatorios. De manera que si el número aleatorio es menor o igual que el índice *Cr* se hereda del mutado, y por otro lado si el número aleatorio es mayor se mantiene del original.
- Durante el proceso de selección, se calcula como en el algoritmo genético la "salud" de cada componente de la población copia y de prueba, y se eligen a los más aptos para formar una nueva población.

5.4 ALGORITMO ABC

El algoritmo ABC está basada en métodos de búsqueda estocásticos[12] con inspiración del comportamiento natural de las abejas comunes en la búsqueda de alimentos. Este tipo de algoritmo presenta ciertas ventajas, por ejemplo, no requiere de gradientes de las funciones objeto; además suele ser ideal tanto para variables discretas como continuas; y también puede proporcionar diferentes soluciones óptimas para un solo problema específico de optimización. Todo esto hace de este algoritmo un método de optimización muy popular y efectivo, ya que se está utilizando en múltiples y diversos campos de investigación.

5.4.1 INTRODUCCIÓN

La primera vez que se habló del algoritmo ABC fue introducido por Karaboga [28] en su trabajo de optimización de problemas numéricos, basados en el comportamiento de búsqueda de alimentos de los enjambres de abejas comunes. En este modelo de algoritmo, las abejas se clasifican en tres tipos:

- Abejas trabajadoras. Son las que han encontrado una fuente de alimentos a explotar.

[12] Un proceso estocástico es aquel cuyo comportamiento es no determinista, es decir, que tras un proceso el subsiguiente estado del sistema está determinado tanto por acciones predecibles como por elementos aleatorios. Por ejemplo, se puede observar procesos estocásticos en el campo de la física, biología, en la inteligencia artificial, etc.

- Abejas observadoras. Son las que esperan en la colmena para recibir información por parte de las trabajadoras sobre la cantidad de néctar que hay en la fuente de alimentos
- Abejas exploradoras. Son las que buscan alrededor de la colmena posibles fuentes de alimentos.

La cual el comportamiento del conjunto es la siguiente, tras obtener el néctar de una fuente de alimentos, la abeja trabajadora vuelve a la colmena para compartir información sobre del lugar donde ha conseguido la comida.

Esto se realiza mediante un baile, ver figura18, si el baile es duradero quiere decir que el lugar de donde viene hay una buena cantidad de recursos, por tanto, el tiempo de durabilidad del baile es proporcional a la calidad de la fuente de néctar. Además, con el baile le indica la posición exacta de esta. De este modo, la abeja observadora, tras la información recibida por parte de las distintas abejas trabajadoras, elige explotar la fuente de recursos más golosa convirtiéndose así de observadora a abeja trabajadora.

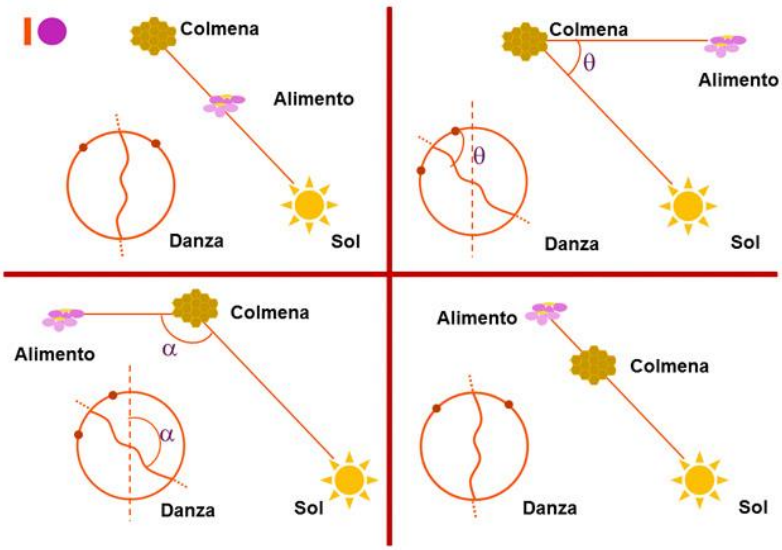


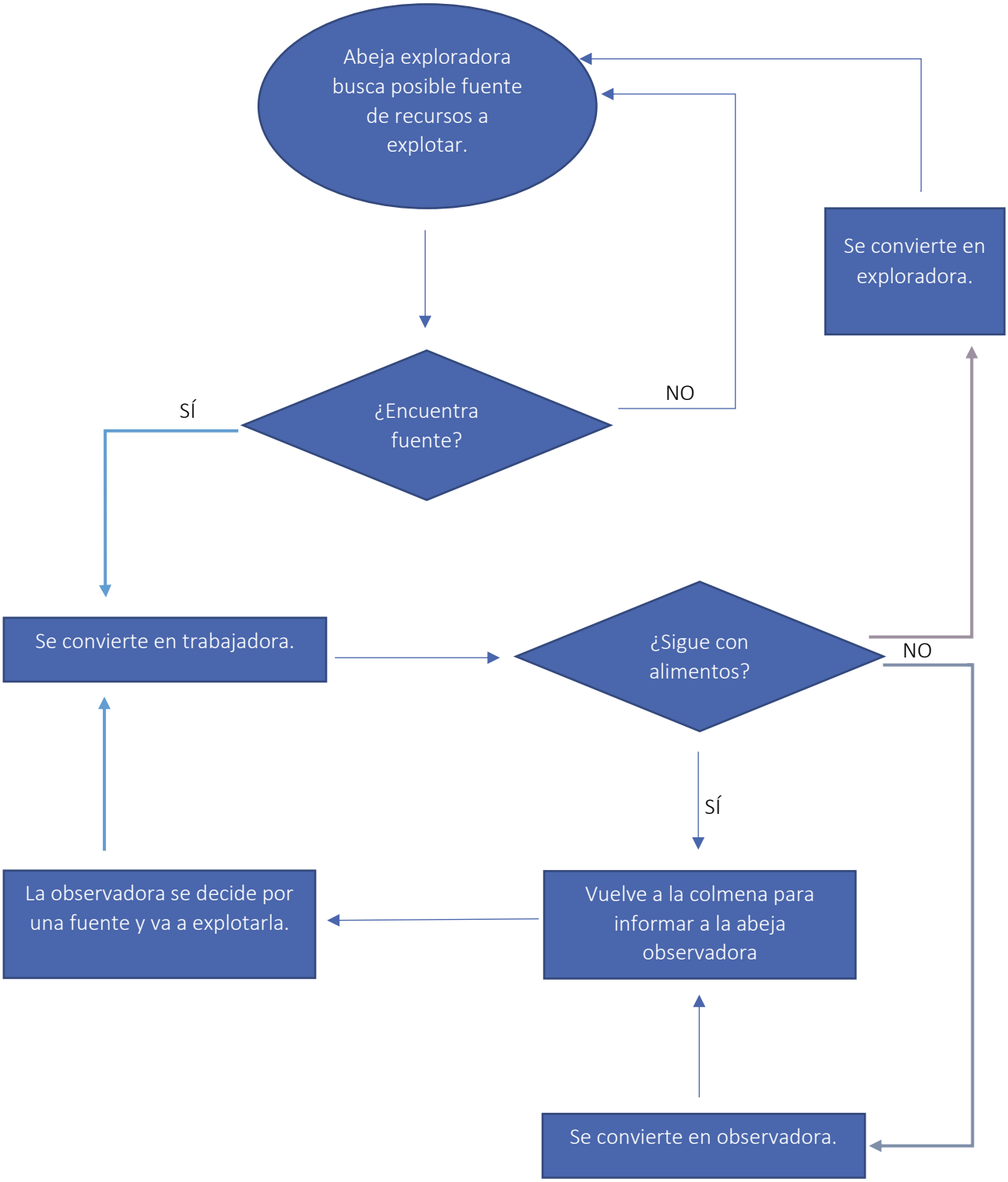
FIGURA 18: ESQUEMA DEL COMPORTAMIENTO DE LAS ABEJAS.

Del mismo modo que las otras dos, si una abeja exploradora encuentra una fuente de alimentos automáticamente se convierte en abeja trabajadora.

Entonces tras explotar por completo una fuente de recursos, todas las abejas trabajadoras abandonan el lugar y se convierten en observadoras y exploradoras.

Resumiendo el comportamiento de estas, se va a ilustrar mediante el diagrama 2 las diferentes etapas y los cambios de los roles de las distintas abejas.

DIAGRAMA 2: FLUJOGRAMA DEL ALGORITMO ABC.



5.4.2 DESCRIPCIÓN DEL ALGORITMO ABC

El diagrama 2 extrapolado a un algoritmo como se ha hecho en el estudio de Ali Hadidi *et al* [29], se debería de realizar de la siguiente manera, mencionando de primera mano que el término ***NS*** se referirá al número de fuentes de alimentos y que será igual al número de abejas trabajadoras u observadoras, además el término ***D*** indicará la dimensión de cada vector solución o se podría decir de otra manera, las coordenadas de la posición de las fuentes de recursos. De este modo los principales pasos para crear el algoritmo serán:

- Primero se debe generar una población inicial aleatoria $(X_1, X_2, \ldots, X_{NS})$, donde cada miembro está compuesta de la siguiente forma $X_i = \{X_{i1}, X_{i2}, \ldots, X_{iD}\}$. Y se utiliza la siguiente expresión para generar cada vector solución (cada miembro de la población).

$$X_{ij} = X_{\min j} + rand\,[0,1] * (X_{\max j} - X_{\min j})$$

  Siendo j = 1, 2,…, D. e i = 1, 2,…, NS.

  Las constantes $X_{min\ j}$ y $X_{max\ j}$ representan el valor mínimo y máximo para cada dimensión j. Entonces, después de generar la población inicial, se debe de evaluar o conocer la "salud" de cada miembro de la población, en este caso en vez de decir la salud, serie más apropiado referirlo como la "la cantidad de néctar" que hay; y cada miembro de población hace referencia a la "posición" en la que se encuentra la fuente de néctar.

- Cada "abeja trabajadora busca", nuevos fuentes de recursos a explotar, alrededor de la fuente de alimentos en la que se encuentra, la cual se utiliza la siguiente ecuación:

$$V_{ij} = X_{ij} + \Phi_{ij} * (X_{ij} - X_{kj})$$

  Donde k e i pertenecen al mismo rango de valores [1, 2,…, NS] pero son distintos, j = 1, 2,…, D y el símbolo Φ representa un valor aleatorio entre [-1, 1] que va cambiando constantemente.

- Después de generar la nueva fuente de alimentos, se evalúa la cantidad de néctar que tienen estos y se procede a la actuación por parte de las abejas trabajadoras la elección de seguir en la posición actual de la fuente o irse a la nueva que se ha encontrado.

Entonces si en la nueva fuente hay más néctar que en la que está ahora se irá a la nueva posición y se incluirá esta posición en la población, en cambio si la nueva que ha encontrado tiene menos que la fuente actual, se quedará en su posición actual.

- El siguiente paso es "informar a las abejas observadoras" lo que han descubierto y elegido para la explotación las abejas trabajadoras, y se hace lo siguiente:

$$p_i = \frac{f_i}{\sum_{i=1}^{NS} f_i}$$

Siendo f_i el valor de la cantidad de néctar que tiene una posición.

Esta ecuación sirve para calcular las probabilidades de selección que tiene cada fuente de recurso. Y se elegirá la que mayor valor tenga. Tras la "elección de la abeja observadora", se generará una nueva posición mediante la primera ecuación y se evaluará esta como en el caso de cuando las abejas trabajadoras encontraban una nueva fuente, si esta nueva fuente generada por las observadoras es mejor que la elegida entonces se reemplazara la posición antigua por esta nueva, en cambio si la nueva generada es peor que la elegida no se hace nada y se mantiene la antigua.

- Si existe una posición de fuente de néctar en la que se ha mantenido durante un buen número de ensayos (como otros algoritmos de optimización, el algoritmo ABC tiene un diseño en la cual se repite el proceso hasta que se cumpla la condición) esta se considerará abandonada y la abeja trabajadora asociada a esta posición se transformará en exploradora, la cual buscará una nueva fuente de la siguiente manera:

$$V_{ij} = X_{\min j} + rand\,[0,1] * (X_{\max j} - X_{\min j})$$

Y se reemplazará por la que se ha abandonado.

- Se repetirá estos pasos hasta que se termine la condición, mostrando al final la mejor posición de las fuentes de néctar que han estado las abejas.

Aquí se concluye la explicación del algoritmo ABC, y basándome en esta descripción he conseguido aplicarlo para la optimización del proceso de búsqueda de las mejores ganancias del controlador PID bilineal.

6 EXPERIMENTOS

6.1 INTRODUCCIÓN

En este apartado se va a contar el procedimiento que se ha seguido para obtener los valores más adecuados para nuestro controlador bilineal PID. Se explicará cada etapa por lo que ha pasado mi trabajo, desde que desarrollé el código del algoritmo ABC hasta que conseguí mis primeras ganancias del controlador.

6.2 ETAPA 1. COMPROBACIÓN DEL ALGORITMO DESARROLLADO. BUSQUEDA DEL MÍNIMO.

Una vez que se ha desarrollado el algoritmo ABC en el programa de Matlab, se comprueba si realmente funciona el código[13].

Para ello, se recurrió a un trabajo que hice durante mi carrera, en concreto a una práctica que hice en la asignatura de *ingeniería de control inteligente*. En esta práctica tuve que buscar el mínimo de una función mediante algoritmos genéticos, así que, en este caso, sustituí el genético por mi algoritmo ABC y, observar si realmente funciona.

La función a estudiar es la siguiente:

$$f(x) = sin^2(3\pi x_1) + (x_1 - 1)^2[1 + sin^2(3\pi x_2)] + (x_2 - 1)^2[1 + sin^2(2\pi x_2)]$$

El cual su mínimo se encuentra cuando las variables equivalen a:

$$[X_1, X_2] = [1, 1]$$

De manera que si se compila y ejecuta mi programa se observa la figura 19:

[13] El código de esta etapa está en el anexo del trabajo.

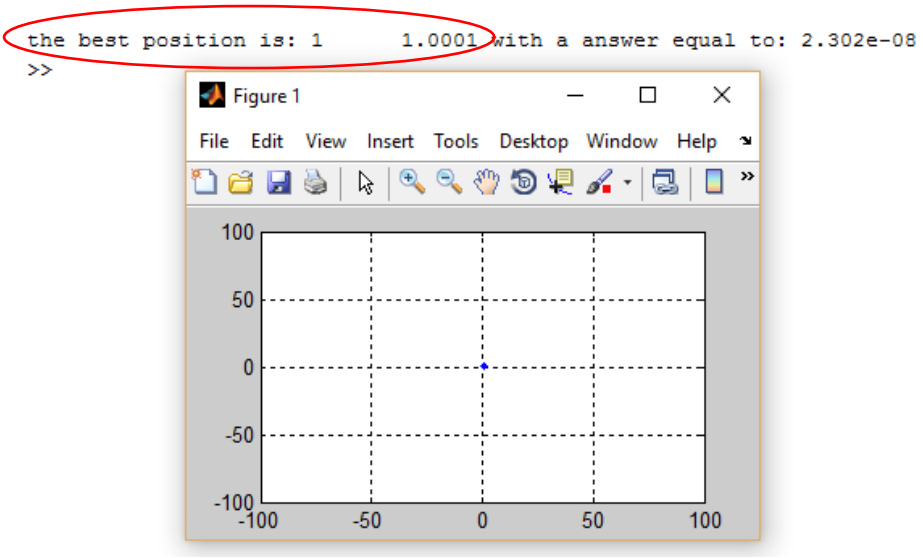


FIGURA 19: RESULTADO EJEMPLO DE LA PRIMERA ETAPA DE LOS EXPERIMENTOS.

Se observa que encuentra correctamente el mínimo de la ecuación, con un error despreciable.

De este modo se concluye esta primera etapa de comprobar si realmente funciona el algoritmo desarrollado. Dando por hecho que sí es así. Por tanto se pasa a un nivel más alto, optimizar el control de un motor.

6.3 ETAPA 2. FIABILIDAD DEL ALGORITMO DESARROLLADO. CONTROL DE UN MOTOR.

En esta etapa se comprueba la efectividad del algoritmo[14] a la hora de optimizar el controlador bilineal PID a través de un modelo de un motor[15], cuya entrada de referencia al motor es un generador de pulsos.

La función de transferencia del motor viene dada por la siguiente expresión:

$$G(s) = \frac{3{,}666}{0{,}2193s^2 + s}$$

[14] El código de esta etapa está en el anexo.
[15] El modelo del motor, diseñado en SIMULINK, ha sido proporcionado por mi tutor del proyecto, Fernando Martín Monar.

Y recordemos que las expresiones del controlador bilineal PID son:

- Por parte del compensador bilineal

$$y\,(k) = \frac{1 + \ k_b \ * y_{ref}(k)}{1 + \ k_b * y(k-1)}$$

- Por parte del PID

$$u(t) = K(e(t) + \frac{1}{t_i}\int_0^T e(t)dt + t_d\frac{de(t)}{dt})$$

De modo que si se compila y ejecuta el programa se observa la figura 20:

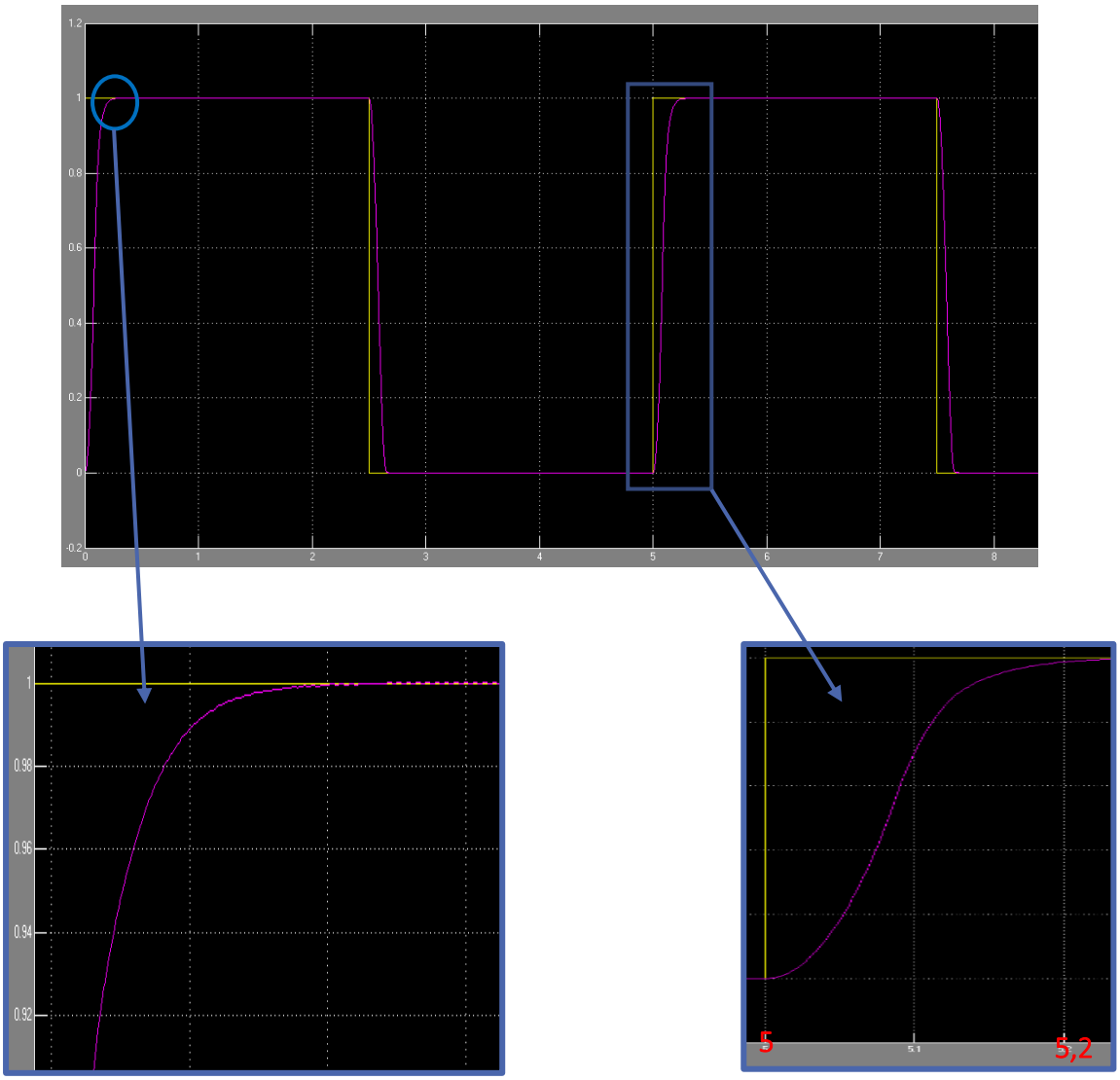


FIGURA 20: RESULTADO EJEMPLO DE LA SEGUNDA ETAPA DE LOS EXPERIMENTOS.

Como se observa en las gráficas, el error es mínimo, no existe sobreoscilación, y la velocidad de subida y de bajada es bastante buena.

```
the best position is: 857.2239      1.176069      30.82379      53.1175 with a answer equal to: 0.028234
full time: 274.0064
```

FIGURA 21: GANANCIAS DEL CONTROLADOR DEL EJEMPLO DE LA FIGURA 19.

En la figura 21 es un recorte de MATLAB indicando la mejor solución que ha obtenido mi algoritmo ABC, cuya interpretación es la siguiente:

- Kp = 857,2239
- Ki = 1,176069
- Kd = 30,82379
- Kb = 53,1175

Con un error cometido de 0,028234. Que es bastante bajo la verdad.

De este modo se da por concluido esta etapa con un buen sabor de boca, ya que nuestro algoritmo ABC funciona[16] adecuadamente como optimizador del controlador bilineal PID.

6.4 ETAPA 3. APLICACIÓN DEL ALGORITMO DESARROLLADO EN EL MODELO SMA.

Finalmente se llega a la etapa de probar el algoritmo[17] en el actuador SMA, y modificando ciertos parámetros[18] para que se adapte al modelo[19] del SMA.

[16] El resultado de los valores de las ganancias para nuestro controlador depende de la configuración del optimizador (el número de ciclos y de muestras, que cuanto mayores son, mayor es la precisión y el tiempo de computo).
[17] El código de esta etapa está en el anexo.
[18] Condicionando de que las ganancias del controlador estén entre [0 1].
[19] El modelo del actuador SMA, un prototipo todavía, se me ha facilitado por mi tutor del proyecto, Fernando Martín Monar.

6.4.1 MODELO SMA EN SIMULINK

En este apartado voy a mencionar brevemente el modelo del actuador SMA en SIMULINK. Este modelo me lo ha proporcionado mi tutor de la práctica, y que ha sido desarrollado por un estudiante de Máter de la Carlos III de Madrid.

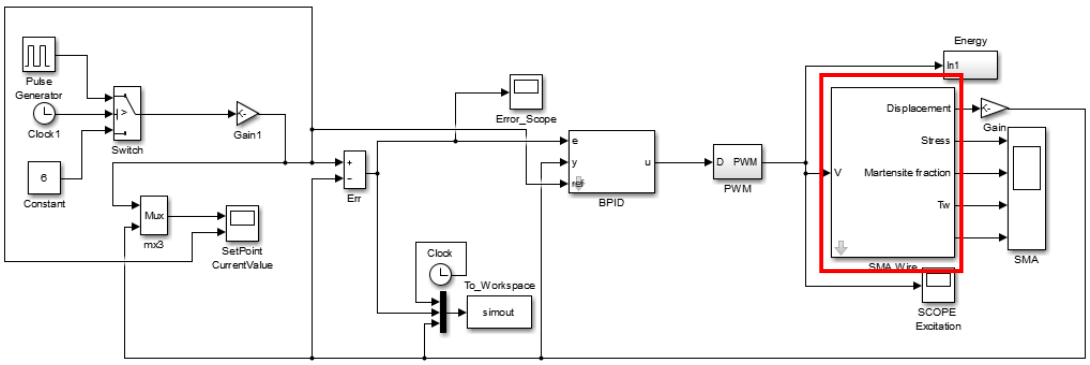


FIGURA 22: MODELO EN SIMULINK DEL ACTUADOR SMA.

Este es el esquema principal del modelo, la cual se observa que los componentes principales son: un generador de pulsos como entrada de referencia, el controlador bilineal PID, el transformador a señal PWM y el modelo del actuador SMA (remarcado por un cuadro rojo). Hay que destacar que es un esquema realmente complejo por el hecho de que el sistema dinámico del SMA es muy complicado, todo ello debido a la histéresis que presenta este.

Entrando en mayor detalle, se puede ver en la figura 23, el desarrollo que tiene el modelo del actuador es el siguiente.

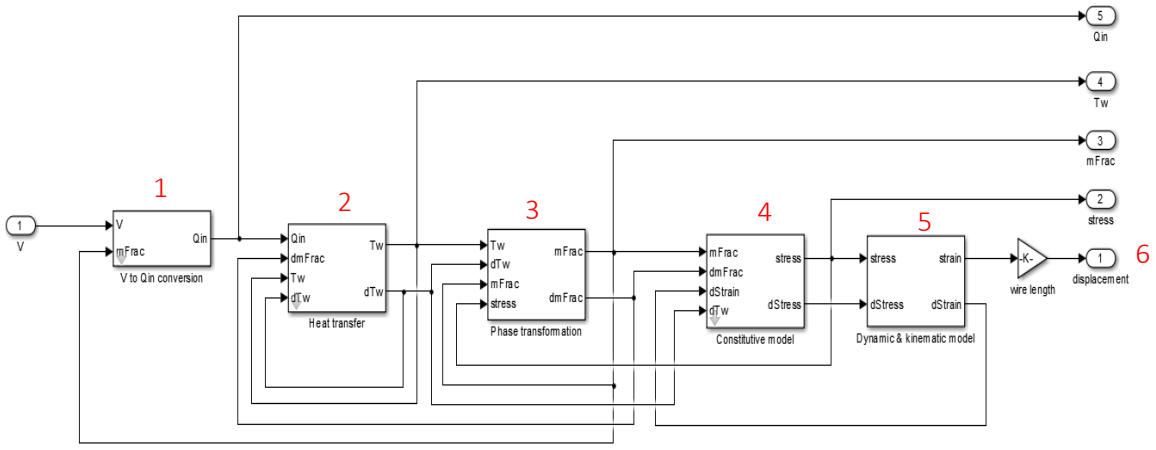


FIGURA 23: CARACTERÍSTICAS DEL SMA. FUENTE: FERNANDO MARTÍN MONAR

La cual para el diseño de este modelo se ha basado en los estudios de los artículos de Brinson *et al* [41] y Ling *et al* [42]. Cuyas fórmulas principales para la correcta reproducción del actuador SMA son las que se mostrarán a continuación y, se seguirá la figura [23] explicando de izquierda a derecha bloque por bloque:

1. Para empezar el primer bloque contiene dos entradas, el voltaje que le tiene que dar la señal PWM según lo que ordene el controlador (*V*) y la información del porcentaje de martensita (*mFrac*), que tiene el SMA en el momento anterior. Y la salida Q_{in} del bloque es la cantidad de energía que se tiene que aplicar al actuador. Por tanto este bloque transforma el voltaje aplicado a energía.

$$Q_{in} = \frac{|V|^2}{[rM * mFrac + rA * (1 - mFrac)] * (\frac{wire\_length}{wire\_section\_area})}$$

   De manera que:

   - $rM = 76 * 10^{-8}$ [Ωm], que es la resistividad de la martensita.
   - $rA = 84 * 10^{-8}$ [Ωm], que es la resistividad de la austenita.
   - $wire\_length = 22 * 10^{-2}$ [m], que es la longitud del cable SMA.
   - $wire\_area\_section = \frac{\pi}{4} * (wire\_diameter)^{-2}$ [m], que es sección del área del cable SMA.
   - $wire\_diameter = 256 * 10^{-6}$ [m], que es el diámetro del cable SMA.

2. Este segundo bloque te permite calcular la transferencia de calor aplicada al actuador SMA a partir de la energía que la aplicamos.

$$Tw = \int_{To}^{T} dTw$$

$$dTw = \frac{Q_{in} - [(h\_SMA + 0{,}01 * Tw\_in^2) * A * (Tw\_in - To) + Aux1]}{wire\_density * V * C}$$

   A continuación se va a indicar que es cada componente:

   - Q_{in}, su valor inicial es de 0. Trabaja en el rango [0 $\frac{9 * wire\_diameter^2}{(256 * 10^{-6})^2}$], si se supera entraría en saturación.
   - $h_{SMA} = 28 \left[\frac{w}{m^2 ℃}\right]$, que es el coeficiente de convección.

- Tw_in, es la temperatura en la que estaba el hilo SMA, en t -1. Su valor inicial es igual a la temperatura ambiente.
- $A = \pi * wire\_diameter\_wire\_length + 2\pi \frac{wire\_diameter^2}{4}$, representa el área superficial del hilo SMA.
- $To = 22℃$, se considera que la temperatura ambiente es de 22 grados centígrados.
- $wire\_density = 6450\ [\frac{Kg}{m^3}]$, es la densidad del hilo SMA.
- $V = \pi * wire\_length * \frac{wire\_diameter^2}{4}$, representa el volumen del hilo.
- $C = 857{,}4\ [\frac{J}{Kg℃}]$, representa el calor específico que tiene el hilo.
- $Aux1 = wire\_density * V * H * Aux2.$
- $H = 2418352\ [\frac{J}{Kg}]$, que representa el calor latente de transformación.
- $Aux2 = \begin{cases} |dmFrac|; if\ dTw \geq 0 \\ -|dmFrac|; if\ dTw < 0 \end{cases}$

3. Este tercer bloque indica la fase de transformación que sufre el hilo al calentarse y enfriarse, indicando la fracción de estructura martensita que hay en el hilo en cada momento.

   Se utiliza el método de diferencial tangente-hiperbólico y posteriormente un integrador para obtener el valor de la fracción de la martensita.
4. Cálculo de la tensión que sufre el actuador SMA.

$$Stress = \int dStress$$

$$dStress = Aux3 + Aux4 + Aux5$$

A continuación se indicará lo que es cada parámetro de la ecuación superior:

- $Aux3 = [Em * mFrac + Ea * (1 - mFrac)] * dStress$, basada en el Módulo de Young, caracteriza el comportamiento elástico del SMA.
  - $Em = 28 * 10^9\ [Pa]$, módulo de elasticidad de la martensita.
  - $Ea = 75 * 10^9\ [Pa]$, módulo de elasticidad de la austenita.
- $Aux4 = Em * TECm * mFrac + Fa * TECa * (1 - mFrac) * dTw$, parámetro para calcular la propiedad termo elástica del material.
  - $TECm = 6{,}6 * 10^{-6}\ [℃^{-1}]$, coeficiente de dilatación de la martensita.
  - $TECa = 11 * 10^{-6}\ [℃^{-1}]$, coeficiente de dilatación de la austenita.

- $Aux5 = -[Em * mFrac + Ea * (1 - mFrac)] * el * dmFrac,$ parámetro para el cálculo de la transformación del material.
- $el = 0{,}035$, indica la máxima tensión recuperable.

5. El último bloque consiste en el cálculo de la deformación que sufre el actuador SMA. Para ello se analiza la dinámica y la cinemática de este.

   En el interior del bloque contiene un conmutador la cual permite calcular la deformación según el modelo dinámico empleado.

   - Si el modelo dinámico sigue la de una masa colgante, la deformación se calcularía de la siguiente forma:

     $$Strain = \frac{1}{s} * (-wire\_length) * (dDs)$$

     Siendo,

     $$dDs = \left[\left[Stress * \left(\pi * \frac{wire\_diameter^2}{4}\right) - C * dDs\right] * \frac{1}{m} - g\right] * \frac{1}{s}$$

   - Si el modelo dinámico sigue a la de un muelle, la deformación estaría expresada de la siguiente forma:

     $$Strain = \frac{1}{s} * (-wire\_length) * \pi * (\frac{wire\_diameter^2}{4}) * \frac{1}{K} * dStress$$

     Siendo en esta caso la constante elástica K = 1000 N/m.

6. Por último está la salida que indica el desplazamiento que ha sufrido el actuador SMA, y se calcula:

   $$Displacement = wire\_length * Strain$$

6.4.2 VALORES DE LAS GANANCIAS DEL CONTROLADOR

Tras interpretar el funcionamiento y el desarrollo del modelo, se procede a utilizar nuestro optimizador para verificar si realmente consigue obtener los resultados deseados. Para ello, se debe modificar mínimamente el código desarrollado para poder compatibilizarlo con el modelo SMA que se me ha proporcionado.

A continuación se muestra, la figura 24, de las numerosas pruebas realizadas, uno de los mejores resultados que he obtenido:

```
the best position is: 0.78466  0.00048922     0.13769     0.068023 with a answer equal to: 85418829.6486
full time: 3829.1383
```

FIGURA 24: RESULTADOS DE LAS GANACIAS DEL CONTROLADOR PID BILINEAL PARA EL SMA.

FUENTE: ELABORACÓN PROPIA.

Con una población igual a 7 y 30 ciclos realizados. Se obtiene la relación de la señal de referencia (señal generador de pulsos) y la controlada (curva de color morado) como se puede apreciar en la figura 25.

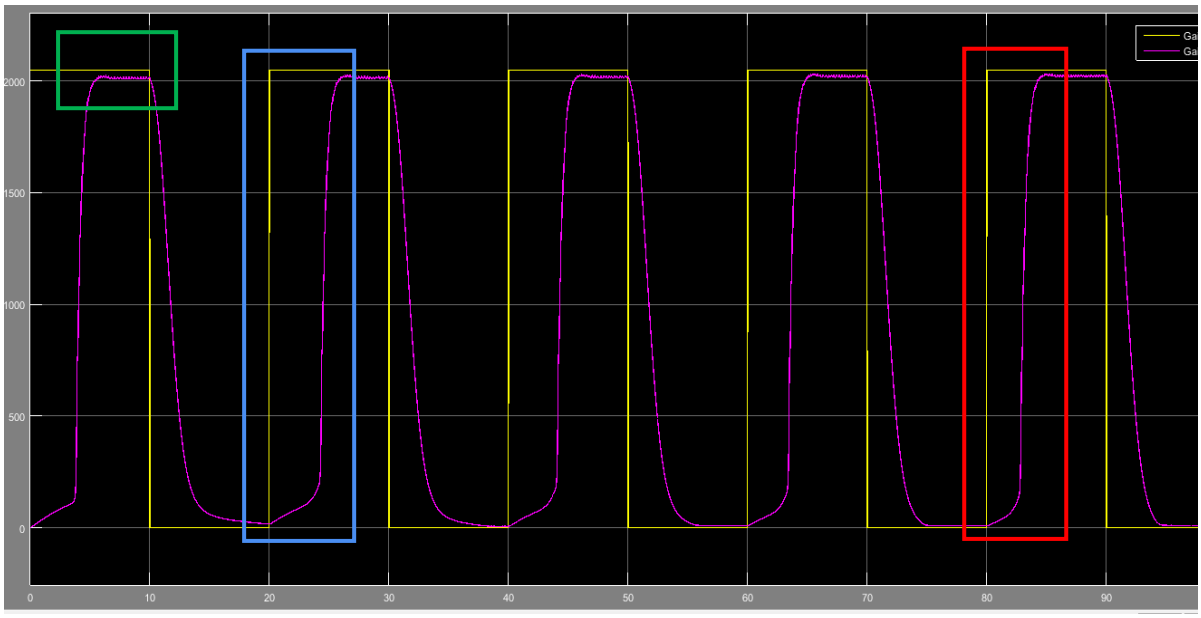


FIGURA 25: RESULTADO EJEMPLO DE LA TERCERA ETAPA DE LOS EXPERIMENTOS.

FUENTE: ELABORACIÓN PROPIA

Por lo que se puede observar en la imagen 25, la respuesta controlada no presenta sobreoscilación (cuadro verde) aunque sí contiene cierto rizado (imagen inferior). Por otro lado, la velocidad de subida parece que va aumentado a medida que pasa el tiempo, ya que podemos apreciar que en el último pulso (cuadro rojo) la respuesta es mucho más rápida que las iniciales (cuadro azul). A continuación se muestra un zoom realizado en la parte superior del último pulso, ver imagen 26.

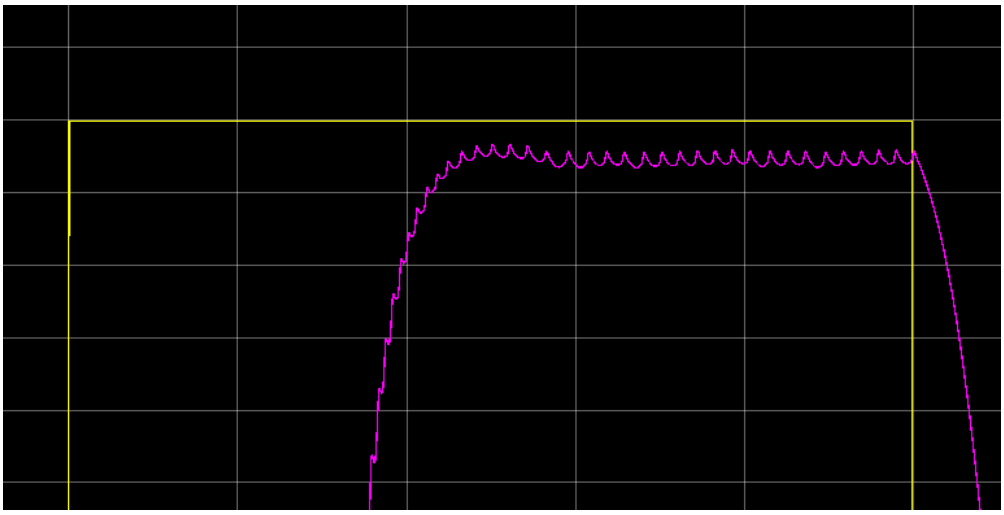


FIGURA 26: ZOOM DEL PRIMER PULSO DE LA FIGURA 24.

El rizado que presenta es de un 0.2%, que en principio es despreciable.

De este modo se considera que se ha obtenido un buen resultado de control, teniendo en cuenta la gran dificultad que hay a la hora de controlar estos tipos de sistemas.

6.5 ETAPA 4. OTRAS PRUEBAS. MODIFICACIÓN DEL ALGORITMO DESARROLLADO.

Tras conseguir que el optimizador halle los valores que más se adecua para controlar el actuador SMA mediante el controlador bilineal PID, se ha modificado el algoritmo inicial como se ha hecho en el trabajo de Ali Hadidi [25], de modo que los cambios que se han realizado son:

- En la fase de la elección de la abeja observadora, en vez de calcular la probabilidad de la cantidad de alimentos que hay en las distintas fuentes y elegir la que mayor tenga, en este caso, se va a utilizar la selección por torneo, que de cara a la programación es mucho más fácil. El funcionamiento de este es el siguiente, se elige aleatoriamente un número de fuentes de alimentos, se calcula la cantidad de néctar de estos y el que mayor cantidad tenga es el ganador del torneo. Entonces, la abeja observadora va a elegir ir a dicha fuente ganadora.

- En la fase de la abeja exploradora, se cambiará de su método de búsqueda la expresión aleatoria por una gaussiana. Ya que la expresión de aleatoriedad es efectiva al comienzo y no tiene sentido utilizarlo a medida que va acercando al resultado deseado. Por tanto, al utilizar la gaussiana (de media igual a cero y una desviación estándar de los D^{20} parámetros de todos los individuos de la población en la iteración de ese instante) la búsqueda pasa de forma global a una búsqueda local.

A continuación se va a mostrar el resultado que se ha obtenido con este nuevo código, figura 27, con el mismo número de población y de ciclos que se ha utilizado para hallar el mejor resultado obtenido en el apartado anterior.

```
the best position is: 0.11466  0.00048355     0.12596     0.95116 with a answer equal to: 76974774.9958
full time: 3185.0363
```

FIGURA 27: RESULTADOS DE LAS GANACIAS DEL CONTROLADOR PID BILINEAL SIENDO MODIFICADO EL ALGORITMO ABC.

Se puede comprobar que al modificar el código se mejora el tiempo de cómputo, además se obtiene un error menor respecto al anterior, ver tabla 1:

TABLA 1: COMPARACIÓN RESULATADOS DE ALGORITMO ABC CON ALGORITMO ABC MODIFICADO.

	Tiempo (s)	**Error**
ABC	3829,1383	85418829,65
Nuevo ABC	3185,0363	76974774,99

A continuación se puede observar la relación de la señal de referencia y la controlada. Puede que parezca que el sistema es peor, pero en este caso se ha llegado a unas ganancias que controlan de una manera más lenta o "suave", de modo que no castiga tanto al actuador. Hay que decir también que por desgracia con estas ganancias la señal controlada no alcanza el valor máximo ni mínimo de referencia.

[20] D es un vector que contiene los valores [Kp Ki Kd Kb], se utiliza en el algoritmo como las coordenadas de la posición de una fuente de alimentos.

También, al igual que en el caso anterior, presenta rizado en todo instante, aunque sea mínimo.

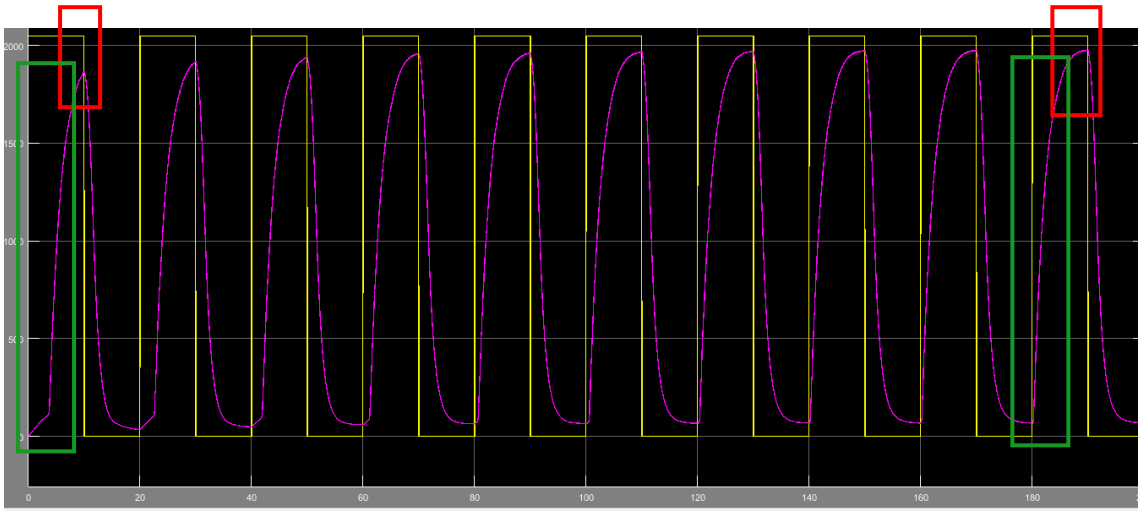


FIGURA 28: RESULATADO EJEMPLO DE LA CUARTA ETAPA DE LOS EXPERIMENTOS.

Se puede comprobar en la figura 28 de que a medida que pasa el tiempo el controlador consigue que el modelo se aproxime cada vez más al máximo de la referencia (cuadros rojos), y además, mejora el tiempo de subida con el tiempo (cuadros verdes). De modo que si comparamos el primer escalón (izquierda) con el último (derecha) se apreciará mejor su control, ver imagen 29.

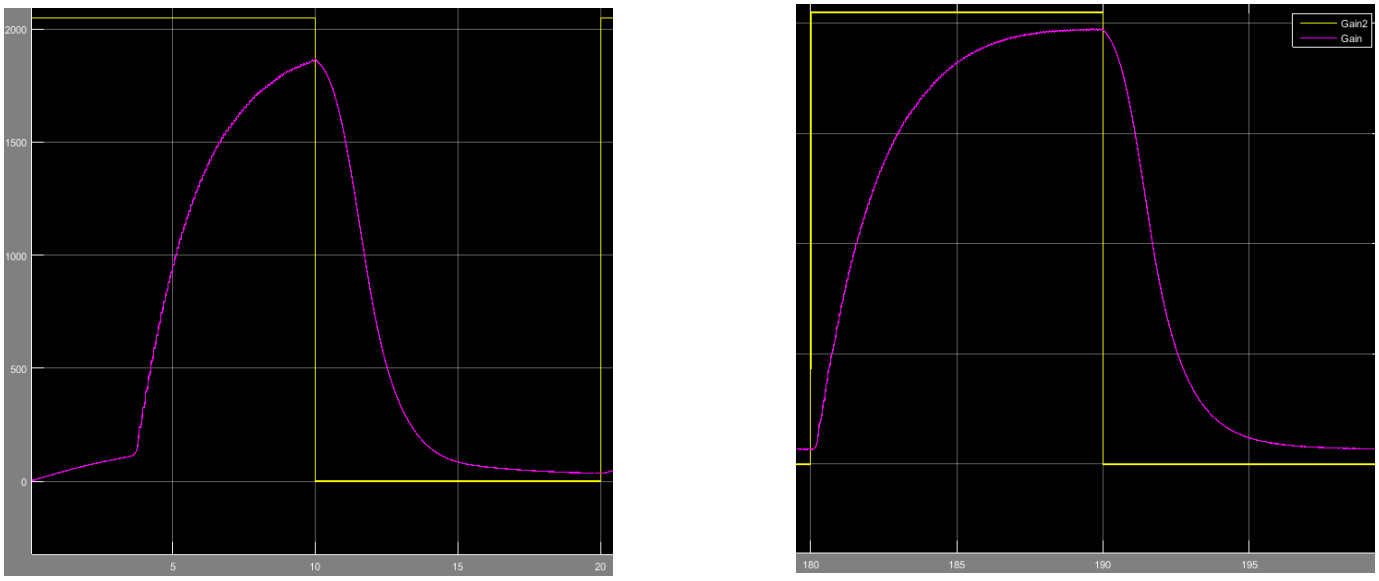


FIGURA 29: ZOOM DEL PRIMER ESCALÓN (IZQ) Y DEL ÚLTIMO ESCALÓN (DCH) DE LA FIGURA 27.

7 CONCLUSIONES

Hemos podido comprobar que el controlador bilineal PID es una buena alternativa para controlar los actuadores de tipo SMA, ya que como se ha visto, la respuesta del modelo en SIMULINK presenta unos resultados realmente bastante buenos. Además, otro punto a favor del controlador bilineal es que es bastante fácil de implementar tanto como en forma de estrategia de control como de compensador de las no-linealidades, como puede ser la histéresis.

También hemos podido ver que los materiales con memoria tienen una gran posibilidad de sustituir en el futuro a los actuadores tradicionales, aunque actualmente esté más enfocado en el uso de la medicina, como materiales para la fabricación de catéteres. Pero es posible que en su día se utilice para mover prótesis o exoesqueletos, eliminando una de las grandes restricciones que hay ahora, que son los grandes tamaños de los actuadores de estos.

Por último, como comentario del algoritmo utilizado para la optimización del control del sistema, hay que decir que es bastante fácil de entender y de aplicar. Ya que al estar basada en el comportamiento de las abejas, en mi opinión, si eres amante de la naturaleza, resulta más fácil y motivador intentar comprender el comportamiento de estas y aplicarlo como algoritmo. Se ha podido ver que para sistemas sencillos se obtienen muy buenos resultados, pero para optimizar el controlador del actuador SMA ya resulta un poco más complicado. Pues requiere de mucho tiempo de para un resultado medianamente aceptable.

En conclusión, el optimizador es sencillo y divertido pero su método de búsqueda queda algo limitada, aunque pueda que el desarrollo de este en código no esté perfecta por mi persona. De todos modos, con el trabajo realizado y los resultados obtenidos, el optimizador cumple con el objetivo, y por tanto estoy más que conforme con el proyecto.

REFERENCIA

[1] S.M. Yang, J.H. Roh, J.H. Han, J. Lee, Experimental studies on active shape control of composite structures using SMA actuators, J. Intell, Master. Syst. Struct. 17 (2006) 767-777.

[2] A.V. Popov, M. Labib, J. Fays, R.M. Botez, Closed-loop control simulations on a morphing wing, J Aircr. 45 (2008) 1749-1803.

[3] http://www.unicauca.edu.co/deic/Documentos/PFC_Andres_Cano_Sanchez.pdf

[4] E. Asua, V. Extebarria, A. García-Arribas, Micropositioning control using shape memory alloys. In: Control Applications 2006. CCA 2006. Proceeding of 2006 IEEE Conference on, Munich, Germany, 2006, pp. 3229-3234.

[5] http://intranet.ceautomatica.es/old/actividades/jornadas/XXVIII/documentos/1771-Jornadas2007.pdf

[6] N. Ma, G. Song, Control of shape memory alloy actuator using pulse with modulation, Smart Master, Struct. 12 (2003) 712-719

[7] G. Song, N. Ma, Control of shape memory alloy actuator using pulse-width-pulse-frequency (PWPF), J. Intell. Master. Syst. Struct. 14 (2003) 15-22.

[8] K.K. Leang, S. Ashley, G. Tchoupo, Iterative and feedback control of hysteresis compensation in SMA. J. Dyn. Syst. Meas. Control 131 (2009), 014502-1-014502-6.

[9] A. Kugami, T.I. Liu, P. Hozian, Control of Shape Memory alloy actuators with a neuro-fuzzy feedforward model element, J. Intell. Manuf. 17 (2006) 45-56.

[10] J. Jayender, R.V. Patel, S. Nikumb, M. Ostojic, Modeling and control of shape memory alloy actuators, IEEE Trans. Control Syst. Technol. 16 (2008) 279-287.

[11] A. Kilicarslan, K. Grigoriadis, G. Song, Compensation of hysteresis in shape memory alloy wire system using linear parameter-varying gain scheduling control. IET Appl. 8 (2014) 1875-1885.

[12] Chang LY, Chen HC. Tuning of fractional PID controllers using adaptive genetic algorithm for active magnetic bearing system. WSEAS Trans Syst 2009;8:158-67.

[13] Zhang J, Zhuang J, Du H, Wang S, Self-organizing genetic algorithm based tuning of PID controllers, Inf Sci 2009; 179:1007-18.

[14] Cao JY, cao BG. Design of fractional order controllers based on particle swarm optimization. In: 1st IEEE conference on industrial electronics and applications; 2006, p. 1-6.

[15] Korani WM, Dorrah Th, Emara HM. Bacterial foraging oriented by particle swarm optimization strategy for PID tuning. In: IEEE international symposium on computational intelligence in robotics and automation (CIRA); 2009. P. 445-50.

[16] Fernando Martín, Concepción A. Monje, Luis Moreno, Carlos Balaguer. DE-based tuning of PID controllers. Robotic Lab, Department of systems Engineering and Automation, Carlos III University, Madrid, Spain. ISA Transactions 59 (2015) 398-407

[17] Dialnet-MaterialesInteligentesAleacionesMetalicasYPolimero-4804628.pdf

[18] http://www.batanga.com/curiosidades/7253/metales-con-memoria-el-nitinol-y-un-truco-que-te-vuela-el-cerebro

[19] http://pressurevesseltech.asmedigitalcollection.asme.org/article.aspx?articleid=1671994

[20] Leppäniemi, A., Shape memory alloys: applications and commercial aspects, [Online] http://www.ad.tut.fi/aci/courses/76527/Seminars2000/SMA.pdf , 2000.]

[21] Mavroidis, C ., Pfeiffer, C ., Michael, M ., Conventional actuators , shape memory alloys and electrorheological fluids, Department of Mechanical and Aerospace Engineering, Rutgers University, NJ, 2000 .

[22] https://adquisiciondedatos.wordpress.com/2013/11/12/introsma/

[23] https://coscomantauni.files.wordpress.com/2014/02/3-estrategias-de-control.pdf

[24] Luis Moreno, Fundamentos de las redes neuronales, Departamento de ingeniería de sistemas y automática, Universidad Carlos III de Madrid, Agosto 2011.

[25] S. Martineau, K.J.Burnham, O.C.L. Haas, G. Andrews, A. Heeley, Four-term bilinear PID controller applied to an industrial furnace, Control Eng. Pract 12 (2004) 457-464.

[26] Álvaro Villoslada, Naiara Escudero, Fernando Martín, Antonio Flores, Cayetano Rivera, Marcelo Collado, Luis Moreno, Position control of a Shape Memory Alloy actuator using a four-term bilinear PID controller, Sensor and actuator A 236 (2015) 257-272.

[27] T. Stützle, Parallelization Strategies for Ant Colony Optimization, Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature, Springer-Verlag, volume 1498, pages 722-731, 1998

[28] Martens, D.; Baesens, B.; Fawcett, T. (2011). «Editorial Survey: Swarm Intelligence for Data Mining». Machine Learning 82 (1): 1-42.

[29] Differential Evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces

[30] D. Karaboga, An idea based on honey bee Swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Enciyes University, Turkey, 2005.

[31] Structural optimization using artificial bee colony algorithm; Ali Hadidi, Sina Kazemzadeh Azad, Saeid Kazemzadeh Azad; Department of Civil Engineering, University of Tabriz, Iran.

[32] "One-Dimensional Constitutive Behavior of Shape Memory Alloys: Thermomechanical Derivations with Non-Constant Material Functions and Redefined Martensite Internal Variable". L.C. Brinson. Journal of Intelligent Material Systems and Structures, 1995. Vol. 4, No. 2, (229-242).

[33] "One-Dimensional Constitutive Relations of Shape Memory Materials". C. Ling, C.A. Rogers. Journal of Intelligent Material Systems and Structures, 1997, Vol. 8, (285-301).

ANEXO 1: PRESUPUESTO

A continuación se presentará el presupuesto que se estima del proyecto. Principalmente se calcula las horas en las que se ha dedicado en realizar dicho trabajo. También se presentará el presupuesto de la dedicación de mi tutor en mi trabajo, por su ayuda y sus consejos.

- Coste personal

TABLA 2: PRESUPUESTO DEL TRABAJO.

Apellidos, Nombre	*Categoría*	*Dedicación (horas al mes)*	*Coste unitario*	*Duración*
Ye Fu, Zhisheng	Ingeniero	75	30	4
			Total:	9.000 €

- Coste por parte de mi tutor

TABLA 3: PRESUPUESTO DEL TRABAJO-2.

Apellidos, Nombre	*Categoría*	*Dedicación (horas al mes)*	*Coste unitario*	*Duración*
Martín Monar, Fernando	Ingeniero	10	40	4
			Total:	1600 €

ANEXO 2: CÓDIGO

En este apartado se va a mostrar el código que se ha tenido que desarrollar para que el optimizador funciones correctamente. Se ha utilizado el programa de MATLAB para la implementación y ejecución de este. También se mostrará una tabla de los resultados que se ha obtenido durante los experimentos.

CÓDIGO DE LA PRIMERA ETAPA

PROGRAMA PRINCIPAL

```
clc
clear all
%*********************************************************************
*****
%
%*********************ABC
ALGORITHM*************************************
%
%*********************************************************************
*****
%Half of the colony are onlookers bees and the other hand are employed
%bees,if the total number of the colony are 200, then 100 are
onlookers and
%the others 100 are employed bees. So:
NS = 7;
%The parameter D is the dimension of the solution vector. Then:
D = 2;
%Now, proceeds to create a random population.
%First, randomly considered to the upper and lower bounds for the
dimension j:
Xmax = zeros(1, D);
Xmin = Xmax;
for r = 1:D
    Xmax(r) = random('unif', 45, 100);
end
for r = 1:D
    Xmin(r) = random('unif', 1, 35);
end
%It is consider the number of trials as: Limit
Limit = 5;
Cont = 0;
Scoutx = 0; %Vaariable to count haow many times a employed bee is
changed to Scout
Cycle = 50; %Number of iterations.
m = zeros(NS, 1);
EvaluateZero = zeros(NS, 3);
PopulationNeighbour = zeros(NS, D);
EvaluateNeighbour = zeros(NS, 3);
```

```
APopulationOnlooker = zeros(1, 2);
PopulationZero = zeros(NS, D);
BestPopulation = Xmax;
for i = 1:NS
    for j = 1:D
        PopulationZero(i, j) = Xmin(j) + random('unif', 0, 1) *
(Xmax(j) - Xmin(j));
    end
end
PopulationLimit = PopulationZero;
%now, the fitness of each food source is evaluated. Is considered the
next
%ecuation to evaluate.
%f(x)= sin^2(3*pi*x1)+(x1-1)^2*[1+sin^2(3*pi*x2)]+(x2-
1)^2*[1+sin^2(2*pi*x2)]
%where, it is looking for the minimum cost. Therefor, is used the
%PopulationZero in the ecuation to looking for the best fitness. The
best
%food source is localizated in the position [1 1].
for o = 1:Cycle
EvaluateZero(:, 1) = PopulationZero(:, 1);
EvaluateZero(:, 2) = PopulationZero(:, 2);
for i = 1:NS
    x1 = PopulationZero(i, 1);
    x2 = PopulationZero(i, 2);
    Ec = Fitness(x1, x2);
    EvaluateZero(i, 3) = Ec;
end
%Until here, is the step 1, initialization of the population and
evaluated
%the fitness of the each food source.
%
%*********************************************************************
*****
%
%Now, the step 2, is create a new food source position. For that, each
%employed bee searches the neighbourhood of its current food source to
%determinate a new food source.
%MY QUESTION IS HOW TO ASSOCIATE THE BEE WITH HIM CURRENT FOOD
%SOURCE POSITION.
for i = 1:NS
    for j = 1:D
        k = randi(NS, 1);
        jota = randi(D, 1);
        PopulationNeighbour(i, j) = PopulationZero(i, j) +
random('unif', -1, 1)*(PopulationZero(i, j) - PopulationZero(k,
jota));
    end
end
%IN THIS STEP, I HAVE CONSIDERED THAT THE RANDOM CHOSE OF THE INDEX j
ONLY IT IS HAPPEN IN THE LAST EXPRESION OF THE ECUATION OF
POPULATIONNEIGHBOUR.
%
%*********************************************************************
*********************************************************************
*********
%
%Step 3, is compare both Population fitness.
EvaluateNeighbour(:, 1) = PopulationNeighbour(:, 1);
EvaluateNeighbour(:, 2) = PopulationNeighbour(:, 2);
for i = 1:NS
```

```
    x1 = PopulationNeighbour(i, 1);
    x2 = PopulationNeighbour(i, 2);
    Ec = Fitness(x1, x2);
    EvaluateNeighbour(i, 3) = Ec;
    if EvaluateNeighbour(i, 3) == 0
        PopulationZero(i, :) = PopulationNeighbour(i, :);
        EvaluateZero(i, 3) = EvaluateNeighbour(i, 3);
    else
        if EvaluateNeighbour(i, 3) < EvaluateZero(i, 3)
            PopulationZero(i, :) = PopulationNeighbour(i, :);
            EvaluateZero(i, 3) = EvaluateNeighbour(i, 3);
        else
            PopulationZero(i, :) = PopulationZero(i, :);
        end
    end
end
%
%*****************************************************************
*****
%
%Step 4, an Onlooker bee select a food source with the information
received
%of all employed bees.
Fi = Onlooker(EvaluateNeighbour(:, 3), NS);
%After selecting a food source, is generated a new food source.
a = EvaluateNeighbour(Fi(2), 1);
b = EvaluateNeighbour(Fi(2), 2);
k = randi(NS, 1);
jota = randi(D, 1);
APopulationOnlooker(1) = a + random('unif', -1, 1)*(a -
PopulationZero(k, jota));
%Is continued using PopulationZero because it is updated in the last
step.
APopulationOnlooker(2) = b + random('unif', -1, 1)*(b -
PopulationZero(k, jota));
Ec = Fitness(APopulationOnlooker(1), APopulationOnlooker(2));
%Now is evaluated, same as the case of employed bees
for i = 1:NS
    if Ec < EvaluateZero(i, 3)
        PopulationZero(i, :) = APopulationOnlooker(1, :);
        EvaluateZero(i, 3) = Ec;
        break;
    else
        PopulationZero(i, :) = PopulationZero(i, :);
    end
end
%End of the step 4
%*****************************************************************
*******
%
%Step 5, If a candidate solution, represented by a food soource be
further
%improved by a predeterminated number of trials, the food source is
%considered abandoned.
Cont = Cont + 1;
if Cont == Limit
    Cont = 0;
    Compare1 = PopulationZero == PopulationLimit;
    for r = 1:NS
        m(r, 1) = mean(Compare1(r, :));
    end
```

```
    indexx = find(m == 1);
    s = size(indexx);
    if s(1) ~= 0;
        for r = 1:s(1)
            for rr = 1:D
                PopulationZero(indexx(r), rr) = Xmin(rr) +
random('unif', 0, 1) * (Xmax(rr) - Xmin(rr));
                Scoutx = Scoutx + 1;
            end
        end
    end
end
EvaluateZero(:, 1) = PopulationZero(:, 1);
EvaluateZero(:, 2) = PopulationZero(:, 2);

%End of the step 5
figure(1)
plot(PopulationZero(:, 1), PopulationZero(:, 2), 'b.', 'MarkerSize',
10);
axis([-100 100 -100 100])
grid on
end
%End of the Step 6
%*****************************************************************
****
%Now, is added diagrams and tables.
TOSORT = sort(EvaluateZero(:, 3));
Position = find(EvaluateZero(:, 3) == TOSORT(1));
for r = 1:D
    BestPopulation(1, r) = PopulationZero(Position, r);
end
disp(['the best position is: ' num2str(BestPopulation) ' with a answer
equal to: ' num2str(TOSORT(1))])
```

PROGRAMAS SECUNDARIOS

- Calculo del coste

```
function Ec = Fitness(x1, x2)
    Ec1 = (sin(3*pi*x1))^2;
    Ec2 = ((x1-1)^2)*(1+(sin(3*pi*x2))^2);
    Ec3 = ((x2-1)^2)*(1+(sin(2*pi*x2))^2);
    Ec = Ec1 + Ec2+ Ec3;
end
```

- Función de las abejas observadoras

```
function Fi = Onlooker(VectorFitness, NS)
SumFitness = 0;
c = 0;
Fi1 = zeros(NS, 1);
for r = 1:NS
    SumFitness = VectorFitness(r) + SumFitness;
end
for r = 1:NS
```

```
        Fi1(r) = VectorFitness(r)/SumFitness;
    end
    c = min(Fi1);
    indexx = find(Fi1 == c);
    Fi = [c; indexx];
    end
```

CÓDIGO DE LA SEGUNDA ETAPA

PROGRAMA PRINCIPAL

```
clc
clear all
%*********************************************************************
*
%   Main Program: ABCBPID.
%   Author: Zhi Sheng Ye Fu.
%   date: July, 2016.
%*********************************************************************
*
%   DESCRIPTION: This script optimizes the parameters of a Bilinear
%                PID in order to control a given plant (defined below
%                by Gp).
%                The Bilinear PID Controller expression is:
%          Bilinear Term : u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
%          PID(s)=Kp + Ki/s + Kd*s
%*********************************************************************
*
%   OUTPUT: Solution:
%                    The best population, time to complete the script
%                    and the best error.
%*********************************************************************
*
%   Transfer function of the process to be controlled.
K=3.666;%35.32;3.4666;%1.357;%20.46;%1.4;%3
T=0.2193;%0.19;0.2193;%0.2693;%0.03573;%0.7;%0.1
Gp=tf(K,[T 1 0]);
%*********************************************************************
*
%   ABC Algorithm variables
Cycle = 40;%Number of iterations.
NS = 10;% Number of particles (in this script, numbers of employed
bees)
D = 4;%Kp, Ki, Kd, Kb
Xmax = zeros(1, D);%Initialization matrix [1x4]
Xmin = Xmax;
for r = 1:D
    Xmax(r) = random('unif', 0.5, 1);
end
%Random maximun and minimum values is generate to use in the equation
%then use it to set up the population.
for r = 1:D
    Xmin(r) = random('unif', 0.0001, 0.5);
end
Limit = 3;%Number of cycles to abandone a posible solution.
%*********************************************************************
*
```

```
%   Some auxiliar variables
PopulationZero = zeros(NS, D);%Initialization matrix [100x4]. It is
%                               a matrix with the possible candidate
%                               solution.
EvaluateZero = zeros(NS, D+1);%Initialization matrix [100x5]. It is
%                               used to visualize the population
%                               (position of the food source) with its
%                               error.
PopulationNeighbour = zeros(NS, D);%Initialization matrix [100x4]. It
%                                    is a matrix with the
%                                    neighbourhood position.
EvaluateNeighbour = zeros(NS, D+1);%Initialization matrix [100x5]. It
%                                    is used to save the parameter
%                                   'PopulationNeighbour' and its
error.
Cont = 0;
Scoutx = 0;%Variable to count how many times a employed bee is changed
%           to Scout
m = zeros(NS, 1);%Initialization matrix [100x1]. It is used to compare
%                  if a possible candidate solution has changed.
APopulationOnlooker = zeros(1, D);%Inicialization matrix [1x4]. It is
%                                   used to generate a new food source
%                                   because a employed bee is changed
%                                   to a scout.
BestPopulation = Xmax;%Initialization matrix [1x4]. This parameter is
%                      used to save the best solution.
%*********************************************************************
*
%   The initial population is generated.
for i = 1:NS
    for j = 1:D
        PopulationZero(i, j) = Xmin(j) + random('unif', 0, 1) *
(Xmax(j) - Xmin(j));
    end
end
%The parameter 'PopulationZero(1, j)' contains 4 values (Kp, Ki, Kd
%and Kb) that in this algorithm represents like the position of
%the food sources for bees.
%*********************************************************************
*
%Contidion for don't create negatives positions
for i = 1:NS
    for j = 1:D
        if PopulationZero(i, j) < 0
            PopulationZero(i, j) = 0.0001;
        else
            PopulationZero(i, j) = PopulationZero(i, j);
        end
    end
end
%*********************************************************************
*
PopulationLimit = PopulationZero;%This is used to compare if a
candidate
%                                  is changed.
tic;%Function to calculate the time os the script.
%*********************************************************************
*
%The begining of the algorithm
for o = 1:Cycle
EvaluateZero(:, 1) = PopulationZero(:, 1);
```

```
EvaluateZero(:, 2) = PopulationZero(:, 2);
EvaluateZero(:, 3) = PopulationZero(:, 3);
EvaluateZero(:, 4) = PopulationZero(:, 4);
%*****************************************************************
*
%Cost function evaluation for the initial population.
for i=1:NS
   EvaluateZero(i, 5)=cost(Gp,PopulationZero(i,1:D));
end
%*****************************************************************
*
%Each employed bee seraches the neighbourhood of its current food
source
for i = 1:NS
    for j = 1:D
        k = randi(NS, 1);
        jota = randi(D, 1);
        PopulationNeighbour(i, j) = PopulationZero(i, j) +
random('unif', -1, 1)*(PopulationZero(i, j) - PopulationZero(k,
jota));
        while PopulationNeighbour(i, j) < 0%This condition is used to
            %                               looking for only positive
            %                               positions.
            PopulationNeighbour(i, j) = PopulationZero(i, j) +
random('unif', -1, 1)*(PopulationZero(i, j) - PopulationZero(k,
jota));
        end
    end
end
%*****************************************************************
EvaluateNeighbour(:, 1) = PopulationNeighbour(:, 1);
EvaluateNeighbour(:, 2) = PopulationNeighbour(:, 2);
EvaluateNeighbour(:, 3) = PopulationNeighbour(:, 3);
EvaluateNeighbour(:, 4) = PopulationNeighbour(:, 4);
%Now, is compared both cost i.e. are the error of the parameter
%'PopulationNighbour' better than 'PopulationZero'?
for i=1:NS
   EvaluateNeighbour(i, 5)=cost(Gp,PopulationNeighbour(i,1:D));
    if EvaluateNeighbour(i, 5) == 0%If the error is null (the best
        %                           error), it is include in the
        %                           parameter 'PopulationZero'.
        PopulationZero(i, :) = PopulationNeighbour(i, :);
        EvaluateZero(i, 5) = EvaluateNeighbour(i, 5);
    else
        if EvaluateNeighbour(i, 5) < EvaluateZero(i, 5)

            PopulationZero(i, :) = PopulationNeighbour(i, :);
            EvaluateZero(i, 5) = EvaluateNeighbour(i, 5);
        else
            PopulationZero(i, :) = PopulationZero(i, :);
        end
    end
end
%*****************************************************************
%This step, consists in see what position is the best and generate
% a new position around it.
Fi = Onlooker(EvaluateNeighbour(:, 5), NS);
%After selecting a food source, is generated a new food source.
a = EvaluateNeighbour(Fi(2), 1);
b = EvaluateNeighbour(Fi(2), 2);
c = EvaluateNeighbour(Fi(2), 3);
```

```
do = EvaluateNeighbour(Fi(2), 4);
k = randi(NS, 1);
jota = randi(D, 1);
APopulationOnlooker(1) = a + random('unif', -1, 1)*(a -
PopulationZero(k, jota));
while APopulationOnlooker(1) < 0%Condition to don't generate negatives
    %                            position.
    APopulationOnlooker(1) = a + random('unif', -1, 1)*(a -
PopulationZero(k, jota));
end
%Is continued using PopulationZero because it is updated in the last
step.
APopulationOnlooker(2) = b + random('unif', -1, 1)*(b -
PopulationZero(k, jota));
while APopulationOnlooker(2) < 0
    APopulationOnlooker(2) = b + random('unif', -1, 1)*(b -
PopulationZero(k, jota));
end
APopulationOnlooker(3) = c + random('unif', -1, 1)*(c -
PopulationZero(k, jota));
while APopulationOnlooker(3) < 0
    APopulationOnlooker(3) = c + random('unif', -1, 1)*(c -
PopulationZero(k, jota));
end
APopulationOnlooker(4) = do + random('unif', -1, 1)*(do -
PopulationZero(k, jota));
while APopulationOnlooker(4) < 0
    APopulationOnlooker(4) = do + random('unif', -1, 1)*(do -
PopulationZero(k, jota));
end
%Now is evaluated its error.
Ec=cost(Gp,APopulationOnlooker);
%And is compared with tha last evaluation.
for i = 1:NS
    if Ec < EvaluateZero(i, 5)
        PopulationZero(i, :) = APopulationOnlooker(1, :);
        EvaluateZero(i, 5) = Ec;
        break;
    else
        PopulationZero(i, :) = PopulationZero(i, :);
    end
end
%**********************************************************************
%Step to see if a food souce cannot be further improved by number of
%trials. If this is the case, that position is changed.
Cont = Cont + 1;
if Cont == Limit
    Cont = 0;
    Compare1 = PopulationZero == PopulationLimit;
    mo = zeros(NS, 1);
    for r = 1:NS
        mo(r, 1) = mean(Compare1(r, :));
    end
    indexx = find(mo == 1);
    so = size(indexx);
    if s(1) ~= 0;
        for r = 1:so(1)
            for rr = 1:D
                PopulationZero(indexx(r), rr) = Xmin(rr) +
random('unif', 0, 1) * (Xmax(rr) - Xmin(rr));
                Scoutx = Scoutx + 1;
```

```
            end
        end
    end
end
%*********************************************************************
%Actualization of thee values.
EvaluateZero(:, 1) = PopulationZero(:, 1);
EvaluateZero(:, 2) = PopulationZero(:, 2);
EvaluateZero(:, 3) = PopulationZero(:, 3);
EvaluateZero(:, 4) = PopulationZero(:, 4);
end
%The end of the algorithm.
%*********************************************************************
Time = toc;% The end of the function tic-toc.
%*********************************************************************
%Loocking for the best position and error
TOSORT = sort(EvaluateZero(:, 5));
Position = find(EvaluateZero(:, 5) == TOSORT(1));
for r = 1:D
    BestPopulation(1, r) = PopulationZero(Position, r);
end
BestError = cost(Gp, BestPopulation);
%*********************************************************************
%Display the information
disp(['the best position is: ' num2str(BestPopulation) ' with a answer
equal to: ' num2str(BestError)])
disp(['full time: ' num2str(Time)])
%*********************************************************************
%The end of the script.
```

PROGAMAS SECUNDARIOS

- Calculo del coste[21]

```
function[error]=cost(Gp,param)
%-------------------------------------------------------------------
--------
%   Function: Cost Function.
%   Author: Fernando Martin Monar.
%   Date: November, 2015
%-------------------------------------------------------------------
--------
% -> Description:
%   It calculates the cost function given a process Gp and a PID
controller.
%   The DE-based scan matching algorithm calls it to estimate the
cost.
%   The Bilinear PID Controller is defined by the following
expression:
%       Blinear Term: u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
%       PID(s)=Kp + Ki/s + Kd*s
%   Several options have been defined for the cost function:
%           1: Time-domain, squared error of the step response.
This
```

[21] Programa diseñado para calcular el coste o error del motor, proporcionado por mi tutor Fernando Martín Monar.

```
%           version runs simulink in order to obtain the error,
taking
%           advantage of the saturation blocks that can be used to
limit
%           the control signal. However, Simulink works slowly.
%           2: Frequency-domain, plane argument in the bode diagram
about
%           the phase margin frequency. Specifications:
%               A_dec; Interval where the phase has to be plane.
%               Wpm>1 && Wpm<50; Limits for the phase margin
frequency.
%               Pm; Minimum Phase Margin
%               M_high_db; High frequencies Magnitude Limit
%               S_low_db; Low frequencies Magnitude Limit
%               Y(35)<0.8; Since the control action cannot be
limited
%               without using simulink, it is limeted by limiting
the rise
%               time. Y is a vector with time resulution and the
component
%               35 is limited to be lower than 0.8
%           3: Mixture between step response and frecuency, giving
%           different weights to each component. (TO BE
IMPLEMENTED)
%-------------------------------------------------------------------
--------
% -> Usage:
%       Inputs: Gp: transfer function of the process to control
%               param = [Kp Ki Kd Kb]; parameters of the BPID
%               option: cost function.
%       Output:  error.
%-------------------------------------------------------------------
--------
% -> See also: Control_BPID nid
%-------------------------------------------------------------------
--------
% Initialization parameters.
warning('OFF');
PHASE=zeros(11,1);
fr_w=0.2;
    Kp=param(1);
    Ki=param(2);
    Kd=param(3);
    Kb=param(4);
    assignin('base', 'Kp', Kp );
    assignin('base', 'Ki', Ki );
    assignin('base', 'Kd', Kd );
    assignin('base', 'Kb', Kb );
    [numG, denG] = tfdata(Gp);
    assignin('base', 'numG', numG );
    assignin('base', 'denG', denG );
    numMatG         = cell2mat(numG);
    denMatG         = cell2mat(denG);
    assignin('base', 'NumGp', numMatG );
    assignin('base', 'DenGp', denMatG );
    sim('SIM_MODEL');                   %Simulacion modelo
    s=size(simout.signals.values);
    m           = s(1,1);                       %Numero de
muestras
    t           = simout.signals.values(1:m,1);   %Tiempo
```

```
    ErrInst     = simout.signals.values(1:m,2);   %Error
InstantÃ¡neo
    d           = simout.signals.values(1:m,3);   %Respuesta
    TempFin     = t(m,1);
    Maximo      = max(d);
    assignin('base', 's', s);
    assignin('base', 'm', m);
    assignin('base', 't', t);
    assignin('base', 'd', d);
    assignin('base', 'ErrInst', ErrInst);
    assignin('base', 'TempFin', TempFin);
    assignin('base', 'Maximo', Maximo);
        ErrInt  = sum(abs(ErrInst));
    error           = ErrInt/m;
    assignin('base', 'error', error);
end
```

- Función de las abejas observadoras

```
function Fi = Onlooker(VectorFitness, NS)
SumFitness = 0;
c = 0;
Fi1 = zeros(NS, 1);
for r = 1:NS
    SumFitness = VectorFitness(r) + SumFitness;
end
for r = 1:NS
    Fi1(r) = VectorFitness(r)/SumFitness;
end
c = min(Fi1);
indexx = find(Fi1 == c);
Fi = [c; indexx];
end
```

CÓDIGO DE LA TERCERA ETAPA

PROGAMA PRINCIPAL

```
clc
clear all
%*************************************************************************
*
%   Main Program: ABCBPID.
%   Author: Zhi Sheng Ye Fu.
%   date: July, 2016.
%*************************************************************************
*
%   DESCRIPTION: This script optimizes the parameters of a Bilinear
%                PID in order to control a SMA actuator.
%                The Bilinear PID Controller expression is:
%          Bilinear Term : u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
%          PID(s)=Kp + Ki/s + Kd*s
%*************************************************************************
*
%   OUTPUT: Solution:
```

```
%                    The best population, time to complete the script
%                    and the best error.
%*************************************************************************
*
%   ABC Algorithm variables
Cycle = 30;%Number of iterations.
NS = 7;% Number of particles (in this script, numbers of employed
bees)
D = 4;%Kp, Ki, Kd, Kb
Xmax = zeros(1, D);%Initialization matrix [1x4]
Xmin = Xmax;
for r = 1:D
    Xmax(r) = random('unif', 0.5, 1);
end
%Random maximun and minimum values is generate to use in the equation
%then use it to set up the population.
for r = 1:D
    Xmin(r) = random('unif', 0.0001, 0.5);
end
Limit = 3;%Number of cycles to abandone a posible solution.
%*************************************************************************
*
%   Some auxiliar variables
PopulationZero = zeros(NS, D);%Initialization matrix [100x4]. It is
%                              a matrix with the possible candidate
%                              solution.
EvaluateZero = zeros(NS, D+1);%Initialization matrix [100x5]. It is
%                              used to visualize the population
%                              (position of the food source) with its
%                              error.
PopulationNeighbour = zeros(NS, D);%Initialization matrix [100x4]. It
%                                   is a matrix with the
%                                   neighbourhood position.
EvaluateNeighbour = zeros(NS, D+1);%Initialization matrix [100x5]. It
%                                   is used to save the parameter
%                                  'PopulationNeighbour' and its
error.
Cont = 0;
Scoutx = 0;%Variable to count how many times a employed bee is changed
%           to Scout
mo = zeros(NS, 1);%Initialization matrix [100x1]. It is used to
compare
%                 if a possible candidate solution has changed.
APopulationOnlooker = zeros(1, D);%Inicialization matrix [1x4]. It is
%                                  used to generate a new food source
%                                  because a employed bee is changed
%                                  to a scout.
BestPopulation = Xmax;%Initialization matrix [1x4]. This parameter is
%                     used to save the best solution.
KpMax = 0.81;
KiMax = 0.0006;
KdMax = 0.08;
KbMax = 1;
%*************************************************************************
*
%   The initial population is generated.
for i = 1:NS
    for j = 1:D
        PopulationZero(i, j) = Xmin(j) + random('unif', 0, 1) *
(Xmax(j) - Xmin(j));
    end
```

```
end
%The parameter 'PopulationZero(1, j)' contains 4 values (Kp, Ki, Kd
%and Kb) that in this algorithm represents like the position of
%the food sources for bees.
%*************************************************************************
*
%Contidion for don't create negatives positions
for i = 1:NS
    for j = 1:D
        if PopulationZero(i, j) < 0
            PopulationZero(i, j) = 0.0001;
        else
            PopulationZero(i, j) = PopulationZero(i, j);
        end
    end
end
%*************************************************************************
*
PopulationLimit = PopulationZero;%This is used to compare if a
candidate
%                                   is changed.
tic;%Function to calculate the time os the script.
%*************************************************************************
*
%The begining of the algorithm
for o = 1:Cycle
EvaluateZero(:, 1) = PopulationZero(:, 1);
EvaluateZero(:, 2) = PopulationZero(:, 2);
EvaluateZero(:, 3) = PopulationZero(:, 3);
EvaluateZero(:, 4) = PopulationZero(:, 4);
%*************************************************************************
*
%Cost function evaluation for the initial population.
for i=1:NS
   EvaluateZero(i, 5)=cost(PopulationZero(i,1:D));
end
%*************************************************************************
*
%Each employed bee seraches the neighbourhood of its current food
source
for i = 1:NS
    for j = 1:D
        k = randi(NS, 1);
        jota = randi(D, 1);
        PopulationNeighbour(i, j) = PopulationZero(i, j) +
random('unif', -1, 1)*(PopulationZero(i, j) - PopulationZero(k,
jota));
        while PopulationNeighbour(i, j) < 0%This condition is used to
looking for only positive positions.
             PopulationNeighbour(i, j) = PopulationZero(i, j) +
random('unif', -1, 1)*(PopulationZero(i, j) - PopulationZero(k,
jota));
        end
        if PopulationNeighbour(i, 1) > KpMax
           PopulationNeighbour(i, 1) = random('unif', 0.5, 0.8);
        end
        if PopulationNeighbour(i, 2) > KiMax
           PopulationNeighbour(i, 2) = random('unif', 0.00045,
0.00059);
        end
        if PopulationNeighbour(i, 3) > KdMax
```

```
            PopulationNeighbour(i, 3) = random('unif', 0.05, 0.14);
        end
        if PopulationNeighbour(i, 4) > KbMax
            PopulationNeighbour(i, 4) = random('unif',  0.8, 1);
        end
    end
end
%**********************************************************************
EvaluateNeighbour(:, 1) = PopulationNeighbour(:, 1);
EvaluateNeighbour(:, 2) = PopulationNeighbour(:, 2);
EvaluateNeighbour(:, 3) = PopulationNeighbour(:, 3);
EvaluateNeighbour(:, 4) = PopulationNeighbour(:, 4);
%Now, is compared both cost i.e. are the error of the parameter
%'PopulationNighbour' better than 'PopulationZero'?
for i=1:NS
   EvaluateNeighbour(i, 5)=cost(PopulationNeighbour(i,1:D));
    if EvaluateNeighbour(i, 5) == 0%If the error is null (the best
        %                           error), it is include in the
        %                           parameter 'PopulationZero'.
        PopulationZero(i, :) = PopulationNeighbour(i, :);
        EvaluateZero(i, 5) = EvaluateNeighbour(i, 5);
    else
        if EvaluateNeighbour(i, 5) < EvaluateZero(i, 5)

            PopulationZero(i, :) = PopulationNeighbour(i, :);
            EvaluateZero(i, 5) = EvaluateNeighbour(i, 5);
        else
            PopulationZero(i, :) = PopulationZero(i, :);
        end
    end
end
%**********************************************************************
%This step, consists in see what position is the best and generate
% a new position around it.
Fi = Onlooker(EvaluateNeighbour(:, 5), NS);
%After selecting a food source, is generated a new food source.
a = EvaluateNeighbour(Fi(2), 1);
b = EvaluateNeighbour(Fi(2), 2);
c = EvaluateNeighbour(Fi(2), 3);
do = EvaluateNeighbour(Fi(2), 4);
k = randi(NS, 1);
jota = randi(D, 1);
APopulationOnlooker(1) = a + random('unif', -1, 1)*(a -
PopulationZero(k, jota));
while APopulationOnlooker(1) < 0 %Condition to don't generate
negatives position.
    APopulationOnlooker(1) = a + random('unif', -KpMax, KpMax)*(a -
PopulationZero(k, jota));
end
if APopulationOnlooker(1) > KpMax
        APopulationOnlooker(1) = random('unif', 0.5, 0.8);
end
%Is continued using PopulationZero because it is updated in the last
step.
APopulationOnlooker(2) = b + random('unif', -1, 1)*(b -
PopulationZero(k, jota));
while APopulationOnlooker(2) < 0
    APopulationOnlooker(2) = b + random('unif', -KiMax, KiMax)*(b -
PopulationZero(k, jota));
end
if APopulationOnlooker(2) > KiMax
```

```
        APopulationOnlooker(2) = random('unif', 0.00045, 0.00059);
end
APopulationOnlooker(3) = c + random('unif', -1, 1)*(c -
PopulationZero(k, jota));
while APopulationOnlooker(3) < 0
    APopulationOnlooker(3) = c + random('unif', -KdMax, KdMax)*(c -
PopulationZero(k, jota));
end
if APopulationOnlooker(3) > KdMax
        APopulationOnlooker(3) = random('unif', 0.05, 0.14);
end
APopulationOnlooker(4) = do + random('unif', -1, 1)*(do -
PopulationZero(k, jota));
while APopulationOnlooker(4) < 0
    APopulationOnlooker(4) = do + random('unif', -KbMax, KbMax)*(do -
PopulationZero(k, jota));
end
if APopulationOnlooker(4) > KiMax
        APopulationOnlooker(4) = random('unif', 0.8, 1);
end
%Now is evaluated its error.
Ec=cost(APopulationOnlooker);
%And is compared with tha last evaluation.
for i = 1:NS
    if Ec < EvaluateZero(i, 5)
        PopulationZero(i, :) = APopulationOnlooker(1, :);
        EvaluateZero(i, 5) = Ec;
        break;
    else
        PopulationZero(i, :) = PopulationZero(i, :);
    end
end
%**************************************************************************
%Step to see if a food souce cannot be further improved by number of
%trials. If this is the case, that position is changed.
Cont = Cont + 1;
if Cont == Limit
    Cont = 0;
    Compare1 = PopulationZero == PopulationLimit;
    for r = 1:NS
        mo(r, 1) = mean(Compare1(r, :));
    end
    indexx = find(mo == 1);
    s = size(indexx);
    if s(1) ~= 0;
        for r = 1:s(1)
            for rr = 1:D
                PopulationZero(indexx(r), rr) = Xmin(rr) +
random('unif', 0, 1) * (Xmax(rr) - Xmin(rr));
                Scoutx = Scoutx + 1;
            end
        end
    end
end
%**************************************************************************
%Condition for max and min values, it is only to support and take
easer
%to looking for the values that I want.
if o < Cycle
    for r = 1:NS
        Condicionante = sum(EvaluateZero(r, :));
```

```
        if Condicionante > 1.9
            PopulationZero(r, 1) = random('unif', 0.75, 0.81);
            PopulationZero(r, 2) = random('unif', 0.00045, 0.00051);
            PopulationZero(r, 3) = random('unif', 0.095, 0.11);
            PopulationZero(r, 4) = random('unif', 0.9, 1.1);
        end
    end
end
%*********************************************************************
%Actualization of the values.
EvaluateZero(:, 1) = PopulationZero(:, 1);
EvaluateZero(:, 2) = PopulationZero(:, 2);
EvaluateZero(:, 3) = PopulationZero(:, 3);
EvaluateZero(:, 4) = PopulationZero(:, 4);

end
%The end of the algorithm.
%*********************************************************************
Time = toc;% The end of the function tic-toc.
%*********************************************************************
%Loocking for the best position and error
TOSORT = sort(EvaluateZero(:, 5));
Position = find(EvaluateZero(:, 5) == TOSORT(1));
for r = 1:D
    BestPopulation(1, r) = PopulationZero(Position, r);
end
BestError = cost(BestPopulation);
%*********************************************************************
%Display the information
disp(['the best position is: ' num2str(BestPopulation) ' with a answer
equal to: ' num2str(BestError)])
disp(['full time: ' num2str(Time)])
%*********************************************************************
%The end of the script.
```

PROGAMAS SECUNDARIOS

- Cálculo de coste[22]

```
function[error]=cost(param)
%-------------------------------------------------------------------
--------
%   Function: Cost Function.
%   Author: Fernando Martin Monar.
%   Date: November, 2015
%-------------------------------------------------------------------
--------
% -> Description:
%   The cost function is, in the time-domain,the squared error of
the step
%   response. This version runs simulink in order to obtain the
error,
%   taking advantage of the saturation blocks that can be used to
limit the
%   control signal. However, Simulink works slowly.
```

[22] Programa diseñado para calcular el coste o error del actuador SMA, proporcionado por mi tutor Fernando Martín Monar.

```
%   It calculates the cost function given a process Gp and a PID
controller.
%   The DE-based scan matching algorithm calls it to estimate the
cost.
%   The Bilinear PID Controller is defined by the following
expression:
%       Blinear Term: u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
%       PID(s)=Kp + Ki/s + Kd*s
%-------------------------------------------------------------------
--------
% -> Usage:
%       Input:  param = [Kp Ki Kd Kb]; parameters of the PID
%       Output:  error. The error is computed as the average
squared error.
%-------------------------------------------------------------------
--------
% -> See also: Control_BPID
%-------------------------------------------------------------------
--------

% Initialization parameters.
warning('OFF');

Kp=param(1);
Ki=param(2);
Kd=param(3);
Kb=param(4);

assignin('base', 'Kp', Kp );
assignin('base', 'Ki', Ki );
assignin('base', 'Kd', Kd );
assignin('base', 'Kb', Kb );

sim('SMA');                    %Simulacion modelo
s           = size(simout.signals.values);
m           = s(1,1);                          %Numero de muestras
t           = simout.signals.values(1:m,1);    %Tiempo
ErrInst     = simout.signals.values(1:m,2);    %Error InstantÃ¡neo
d           = simout.signals.values(1:m,3);    %Respuesta
TempFin     = t(m,1);
Maximo      = max(d);
assignin('base', 's', s);
assignin('base', 'm', m);
assignin('base', 't', t);
assignin('base', 'd', d);
assignin('base', 'ErrInst', ErrInst);
assignin('base', 'TempFin', TempFin);
assignin('base', 'Maximo', Maximo);

ErrInt  = sum( abs(ErrInst) );
error           = ErrInt;
assignin('base', 'error', error);
end
```

- Función de las abejas observadoras

```
function Fi = Onlooker(VectorFitness, NS)
SumFitness = 0;
```

```
c = 0;
Fi1 = zeros(NS, 1);
for r = 1:NS
    SumFitness = VectorFitness(r) + SumFitness;
end
for r = 1:NS
    Fi1(r) = VectorFitness(r)/SumFitness;
end
c = min(Fi1);
indexx = find(Fi1 == c);
Fi = [c; indexx];
end
```

CÓDIGO DE LA CUARTA ETAPA

- En el caso de utilizar el método de selección por torneo, me permite ahorrar el uso de una función auxiliar para que la abeja observadora elija una fuente de alimentos. Por tanto, la función de las abejas observadoras queda eliminada en esta etapa, y el fragmento de código que le sustituye estará en el programa principal con la siguiente forma.

```
Aleatorios = RandiSinRepetir(NS, M);
for r = 1:M
    ParticipantesT(r, 1) = EvaluateNeighbour(Aleatorios(r), 3);
end
ParticipanteG = min(ParticipantesT);
indexxPG = find(EvaluateNeighbour(:,3) == ParticipanteG);
a = EvaluateNeighbour(indexxPG, 1);
b = EvaluateNeighbour(indexxPG, 2);
c = EvaluateNeighbour(indexxPG, 3);
d = EvaluateNeighbour(indexxPG, 4);
```

En cambio he tenido que añadir una función para que genere números aleatorios sin repetir para elegir el mejor de ellos:

```
function AleSR = RandiSinRepetir(M,K)
n = 0; % contador de # aleatorios
AleSR = M-1;
while (n < K)
a = randi(M,1);
    if ((a == AleSR) == 0)
    AleSR = [AleSR, a];
    n = n+1;
    end
end
AleSR = AleSR(:,2:end)';
end
```

- La segunda modificación, el uso de una distribución normal gaussiana en vez de la búsqueda aleatoria, consiste en sustituir:

```
PopulationZero(indexx(r), rr) = Xmin(rr) + random('unif', 0, 1)
* (Xmax(rr) - Xmin(rr));
```

Por el siguiente fragmento:

```
for jj = 1:D
       desvi(jj) = std(PopulationZero(:, jj));
    end
    for ii = 1:D
        PopulationScout = PopulationZero(NS, ii) + normrnd(0,
desvi(ii));
        PopulationZero(NS, ii) = PopulationScout;
    end
```

TABLAS DE RESULTADOS

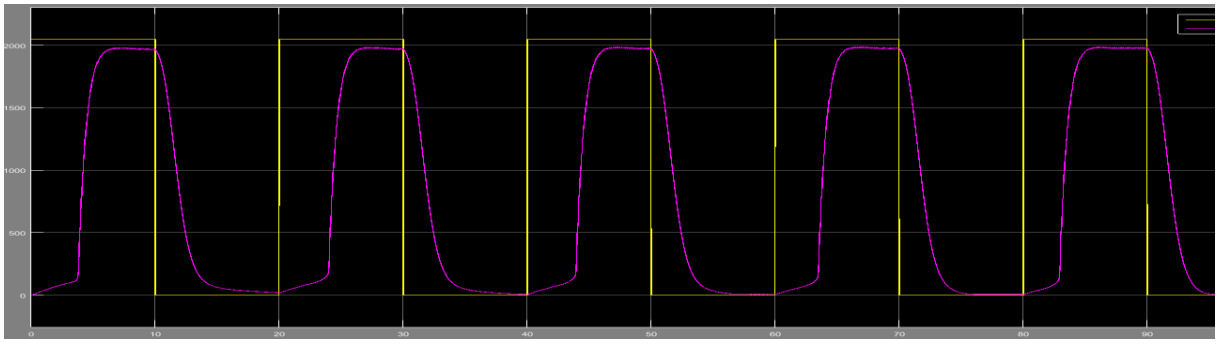
A continuación se va a mostrar una serie de valores que se ha obtenido durante los experimentos. La cual se ha estado modificando ciertos parámetros[23] para observar el comportamiento del modelo SMA ante una entrada de generadores de pulsos.

- **NS = 7 con CICLE = 15**

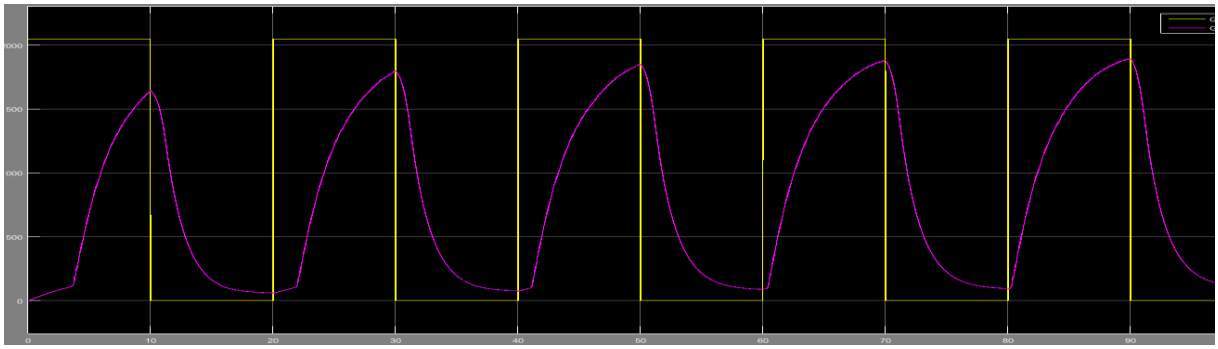
Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	1,1477	0,000508	0,33373	0,98	1359,8
2	0,3538	0,002093	0,59876	0,8892	1456,9

[23] Los resultados se obtienen modificando en el código los parámetros NS y CICLE, que respectivamente son: el número de población, es decir el número de abejas, y las veces que se ha utilizado el algoritmo, ya que como todo optimizador evolutivo, necesita un número de ciclos para hallar el mejor resultado. Todas las pruebas se realizaron con el programa de la etapa 3.

- Prueba 1

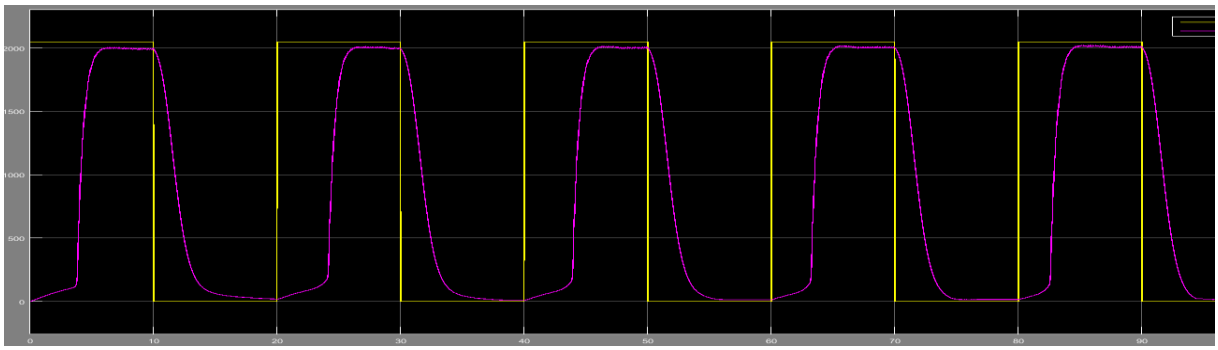


- Prueba 2



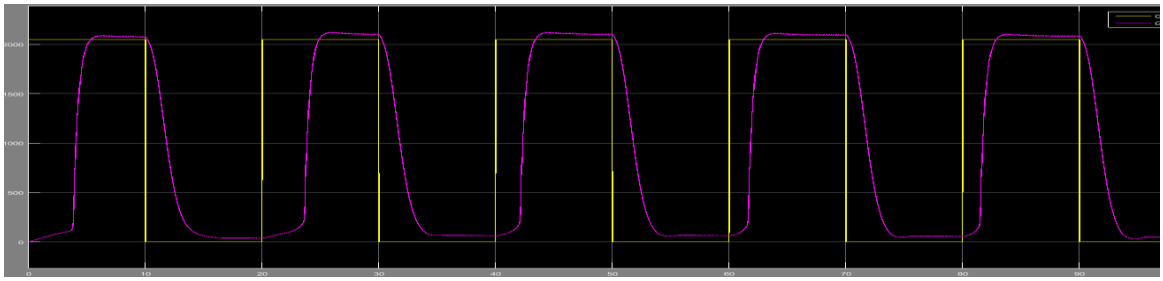
- **NS = 7 con CICLE = 23**

Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	1,407	0,00137	0,32198	0,9175	2412,3



- **NS = 7 con CICLE = 27**

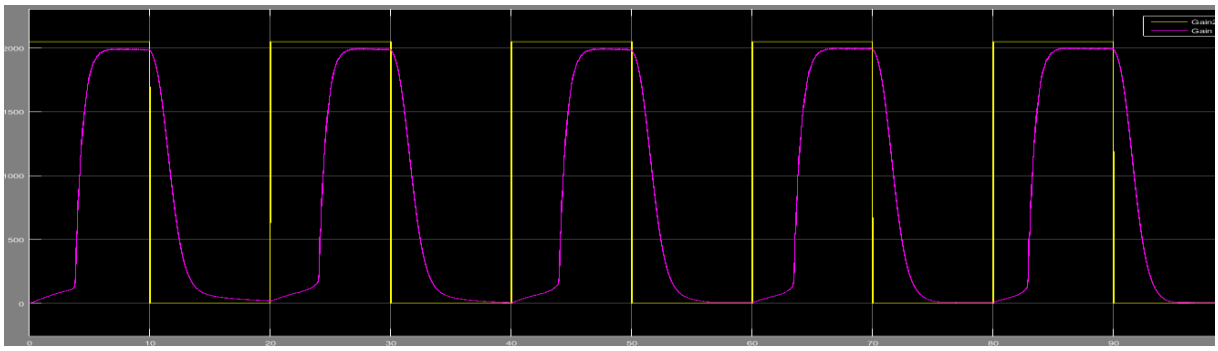
Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	0,99338	0,00943	0,1872	0,93085	2574,5



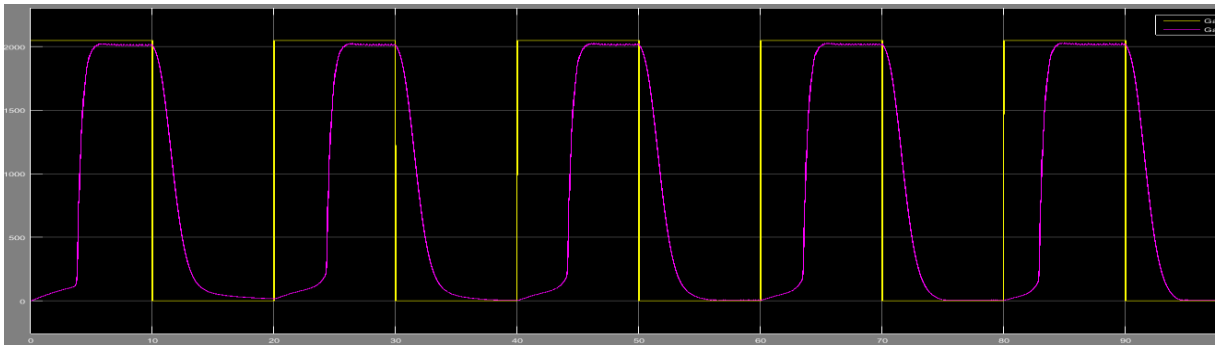
- **NS = 7 con CICLE = 32**

Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	0.95476	0,000487	0,26362	1,4033	2009
2	1,9433	0,000739	0,32309	0,93493	2986,75

  - Prueba 1

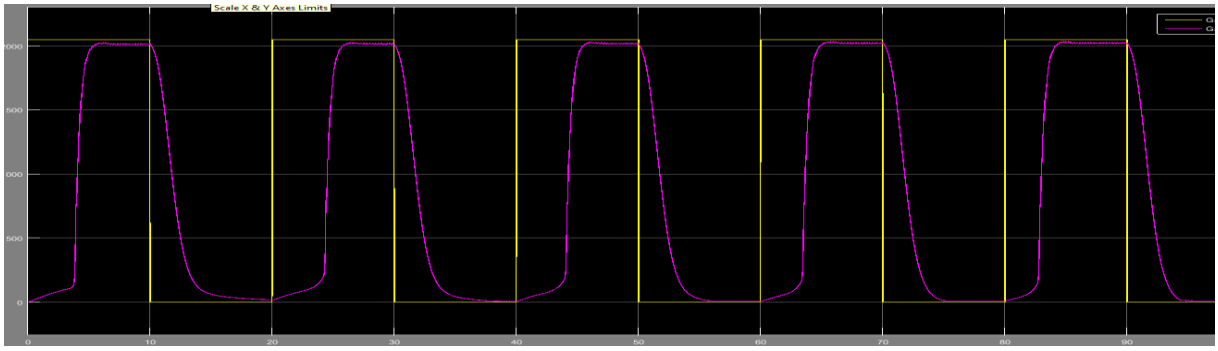


  - Prueba 2



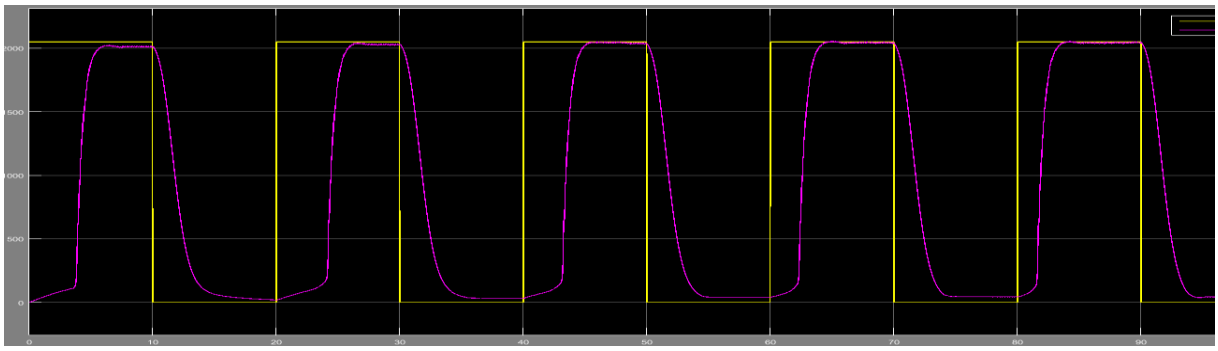
- **NS = 7 con CICLE = 30**

Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	0,79466	0,000489	0,13769	0,068023	3829,14



- **NS = 10 con CICLE = 32**

Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	0,58048	0,001955	0,13551	0,43506	4331,83



- **NS = 10 con CICLE = 40**

Prueba	*Kp*	*Ki*	*Kd*	*Kb*	*Tiempo(s)*
1	0,95802	0,000526	0,1292	0,9396	7522